UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE GRAU

TFG TITLE: Automation of the determination of Inertia Tensors of CubeSats using torsion pendulum period measurements

DEGREE: Grau en Enginyeria de Sistemes Aeroespacials

AUTHOR: Jon Gutiérrez Ibarra

ADVISORS: Pilar Gil Pons
Jordi Gutiérrez Cabello

DATE: July 26, 2020

Título: Automatització de la determinació del tensor d'inèrcia d'un CubeSat a partir de mesures del periode d'un pèndol de torsió.
Autor: Jon Gutiérrez Ibarra

Directores: Pilar Gil Pons
Jordi Gutiérrez Cabello

Fecha: 26 de julio de 2020

Resumen

Context: CubeSats constitute the most extended standard of small satellites, whose presentation in the late 1990's revolutionized the space industry. Their limited, size, the implementation of off-the-self components and consequently, their low-cost, opened possibilities in space exploration and exploitation to academic institutions, to the private sector, and to developing countries.
Aims: As in the case of larger standard satellites, attitude control is critical for the success of many *CubeSat* missions. Specifically, attitude control requires a good knowledge of the inertia tensor of the satellite. In this work, we use the preliminary design of a torsion pendulum aimed to the measurement of the inertia tensor of *CubeSats*. We develop a code which will use, as inputs, specific configurations or positions of a CubeSat in the pendulum, as well as the corresponding period measurements. The outputs of our code will be the position of the centre of mass, the associated moments of inertia and, ultimately, the inertia tensor of the *CubeSat* we are interested in (in the desired frame of reference), together with the calculated precision of our output data.
Methods: We will apply basic concepts of rotational mechanics, namely, the concept of inertia tensor, rotation matrices and the Parallel Axes Theorem, as well as Euler equations for rotation. This information, together with that related to the design of the existing torsion pendulum, will be implemented in a MATLAB code.
Results: A simultaneously flexible and user friendly MATLAB code, which meets the original purpose, was developed and thoroughly described. The code also includes additional features, mostly related to the computation of simulated experiments, and to error calculations. It can be used to refine the original design of the torsion pendulum, and it will be made available for future users of the pendulum. The resulting report will serve the purpose of a users' manual, which will facilitate the interpretation of period measurements, and the coherence in tensor of inertia derivations and results.

Title: Automation of the determination of Inertia Tensors of CubeSats using torsion pendulum period measurements

Author: Jon Gutiérrez Ibarra

Advisors: Pilar Gil Pons
Jordi Gutiérrez Cabello

Date: 26 de julio de 2020

Overview

Contexto: Los *CubeSats* constituyen el estándar de satélites pequeños más extendido, prácticamente desde que fueron presentados a finales de la década de 1990. Los *CubeSats* revolucionaron la industria espacial. Gracias a su pequeño tamaño y por el hecho implementar componentes comercializados, su coste es muy bajo, y con ello abren posibilidades de exploración espacial a instituciones educativas, al sector privado y a países en vías de desarrollo.
Objetivos: Tal y como ocurre con satélites de mayor tamaño, el control de actitud puede resultar crítico para el éxito de su misión, y dicho control de actitud requiere un buen conocimiento del tensor de inercia del satélite. En este trabajo, partimos del diseño preliminar de un péndulo de torsión destinado a obtener tensores de inercia de *CubeSats*. Nuestro objetivo es desarrollr un código que utilize como datos de entrada las configuraciones específicas que le demos al *CubeSat* situado sobre el péndulo, así como las correspondientes medidas de periodo de oscilación. Como información de salida ofrece la posición del centro de masas del péndulo, sus momentos de inercia y, finalmente, el tensor de inercia completo para el sistema de referencia deseado y la precisión de los datos de salida. El programa desarrollado se puede utilizar para mejorar el diseño del péndulo original.
Métodos: Aplicaremos conceptos y leyes básicos de mecánica de rotación (e.g. centro de masas, tensor de inercia, teorema de ejes paralelos, matrices de rotación, ecuaciones de Euler para rotación). Dicha información y el diseño del péndulo de torsión propuesto en un trabajo anterior serán utilizados para implementar un código en MATLAB.
Resultados: El principal resultado de nuestro trabajo es el programa en MATLAB, flexible en cuanto opciones de uso y a la vez sencillo para el usuario. El programa cumple todos los requisitos inicialmente propuestos, e incluso otros adicionales relacionados, sobre todo, con la posibilidad de simular experimentos y con el cálculo de errores en los datos de salida. Se podrá utilizar para refinar el diseño del péndulo inicialmente propuesto. Además, estará disponible para futuros usuarios del péndulo de torsión, una vez construido, junto con la correspondiente memoria, que se espera que sirva como manual de instrucciones detallado. Todo ello facilitará la interpretación de las medidas y cálculos, y la coherencia en la derivación de los tensores de inercia.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author of this degree thesis wants to place on record his sincere gratitude to Jordi Gutierrez and Pilar Gil, without whom this project wouldn't be possible, for giving me the chance to work with them and learn from this fascinating field, and their committed work in this project.

Finally, I take this opportunity to thank my family for their support and encouragement throughout this degree thesis and helping me become an aerospace engineer.

CHAPTER 1. INTRODUCTION

For the last 30 years Space has become an increasingly popular endeavour beyond the activities carried out by standard space exploration agencies, extending to, more recently, within educational institutions and privateers. The reason for this diversity in agents involved in space activities is mainly associated to the development of small and relatively cheap satellites (*small satellites*), which make space accessible to teams with limited practical experience counting with low and moderate budgets. As the number of satellites orbiting the Earth and their applications rise, the means to accurately predict, measure and change their behaviour in orbit become increasingly important. One of the main aspects of a satellite's state and operation is its attitude, which refers to the orientation of an aerospace vehicle with respect to a certain frame of reference. An accurate attitude determination depends on a good knowledge of the inertia tensor of the aforementioned vehicle. In this work, we improve the mathematical approach and develop the software required by the experimental setup proposed in a previous TFG project (1), which was aimed to the experimental determination of inertia tensors of small satellites.

1.1. Context and motivation

The *CubeSat* standard was originally defined in the 1990's (2) as a way to give access to space to universities, with the specific goal of becoming an affordable option to develop various space-related learning projects. CubeSats provide a simple and cheap platform, with shorter construction and design timescales with respect to common satellites.

The *CubeSat* standard specifies that the satellite must be composed of one or more defined cubic units. Each of these units must be a box with a size of $10\times10\times11.3$ cm^3 and a weight of no more than 1.33 kg. A certain number n of these cubic units can be stacked on top of each other resulting in a *nU CubeSat* with a base of 10×10cm^2. *CubeSats* can be launched from a common deployment system such as the Poly-Picosatellite Orbital Deployer (P-POD). This deployment system is compatible with practically all launch vectors, and is usually carried as a secondary payload. The use of standard deployment systems allows to avoid the need of designing and qualifying an interface for the launcher, thus reducing the complexity of each mission.

Following the low-cost approach, a significant number of *CubeSats* make use of commercial off-the-shell components for their electronics, structure and various other systems. The use of components are already developed and proven further reduces the cost, and provides a way to build *CubeSats* without the need to create and certify a system from the ground up. As a consequence of the high demand for *CubeSat* components sever al companies now offer products specifically designed to be used in *CubeSats*. These items can range from power sources and propulsion, to software and communication systems. Furthermore there are some companies which already offer a fully designed and built *CubeSat* off the shelf like the *ArduSat*, that incorporates *Arduino* technology in a *CubeSat*.

Although originally developed for academic purposes several agencies and privateers have seen the potential of *CubeSats*, and have made use of them for their own purposes. Companies such as Spire which currently has a constellation of over 80 *CubeSats*. Spire employs them for acquiring global data sets that include weather patterns and air traffic tracking.

Figure 1.1: *Bird CubeSats* released in low Earth orbit. (Image credit: NASA).

Agencies such as NASA and ESA have put *CubeSats* at the forefront of exploration in missions like the *InSight* lander(2018). The aim of this mission was to place a stationary lander on the surface of Mars, in addition to the *InSight* two identical 6U *CubeSats* called *MarCO* were also launched alongside the main spacecraft. The purpose of these auxiliary satellites was to serve as a communication relay between Earth and the lander in the entry, descent and landing (EDL) phase of the flight.

Because of their intrinsic simplicity, it is not always viable to modify small satellites' behaviour once in orbit. This means that the physical properties of the satellite in question, such as its mass, centre of mass and, in our case, its inertia tensor, must be well determined in advance. Frequently, CAD programs are used to determine the inertia tensor in *CubeSats*. This software models the physical properties of the components and provide the total inertia tensor with respect to a known frame of reference. However, this result is subject to the limitations of modelling assumptions and simplifications (for instance related to wiring and precise component weight and position variations). In practice, any discrepancy between the real satellite ensemble and the simulated one can cause significant deviations between numerical results and real values of the inertia tensor components. As attitude control can be a vital aspect in order to complete a satellite's mission, the need arises for an experimental method to precisely measure the inertia tensor of the actual built satellite.

1.2. Previous work

As *CubeSat* applications increase, so does the demand for reliable methods to calculate their inertia tensor. A previous TFG (1) was devoted to the design of a experimental setup based on a torsion pendulum, which could allow precise measurements of inertia moments of a *CubeSat* and, ultimately, the determination of its entire inertia tensor. The main components of the torsion pendulum consist of a fixed platform and a moving platform connected through a torsion fiber. The moving platform, on top of which the *CubeSat* is to be placed, is able to oscillate with respect to the fixed one, which holds the sensor which

will eventually measure the oscillation period. The physical foundation of this experimental setup is the relation between the oscillation period of the pendulum, and the inertia moment of the *CubeSat* with respect to an axis parallel to the torsion fiber. Steiner's theorem and rotation matrix algebra allow to build the entire inertia tensor after consecutive measurements of the moment of inertia of the *CubeSat* placed in different positions and orientations.

1.3. Goals of this project

In this work we refine the mathematical details of the experimental setup proposed in the TFG project (1), which was aimed to the experimental determination of inertia tensors of small satellites. We develop a purpose-built software aimed to test the experimental design, to quantify the important effects of measurement errors, and specially, to help as a guideline to the users of the experimental setup, once it is built.

1.4. Project structure

This work is structured as follows.

- Chapter 2 summarizes all the theoretical analysis required to understand a damped torsion pendulum, the basics of rotation dynamics, and the components of the inertia tensor of a given rigid solid.
- Once the theoretical fundamentals are described, a brief explanation of the experimental setup on which this work is based will be presented in Chapter 3. In this chapter we also explain the theoretical details of the procedure which will allow to compute the necessary inertia tensor data using the measured periods of the pendulum, on the basis of the theoretical fundamentals of Chapter 2. The different steps followed will be explained together with simplified sketches depicting the different configurations of the experimental setup.
- In Chapter 4 the *Matlab* code developed will be reviewed and tested. We will present all the steps and descriptions of the additional simulation capabilities of the program.
- To sum up, in Chapter 5 we discuss the main results of this thesis, as well as possible improvements to the program and future tasks that would be required for an efficient use of this facility.

CHAPTER 2. THEORETICAL PRINCIPLES

In this chapter the physical magnitudes and principles that describe rotation will be summarized. Specifically, the meaning of the inertia tensor will be explained, and Euler equations for rotation will be enunciated. Then we will briefly describe the physics of the torsion pendulum, and relate it to the concept of inertia tensor. The information for this chapter was extracted from references (3) and (4).

Our analysis is referred to the rotation of rigid bodies, that is, to objects whose relative distance between any arbitrary particles remains constant.

2.1. Main concepts of 3D rotation

2.1.1. Angular momentum and inertia tensor for a system of particles

Much like the linear momentum, the angular momentum is a physical magnitude associated to the momentum of a body in motion. Linear momentum represents this magnitude for a particle of mass m_i undergoing a translation $\vec{p}_i = m_i\vec{v}_i$, where $\vec{p}_i$ is the linear momentum itself, and $\vec{v}_i$ is the particle's velocity. Angular momentum is the analogous magnitude for a body in rotation, and is referred to a certain point in space O (typically the origin of coordinates of a given reference frame, or the centre of mass of the system). The angular momentum of a particle of mass m_i, $\vec{L}_i$, can be expressed as $\vec{L}_i = m_i\vec{r}_i \times \vec{v}_i$, where m_i and $\vec{v}_i$ have the same meaning as above, and $\vec{r}_i$ is the position vector of the particle with respect to the point for which $\vec{L}_i$ is calculated. If instead of referring to a particle, the angular momentum $\vec{L}$ refers to an entire system of N particles of mass m_i each, then $\vec{L} = \sum_{i=1}^{N} m_i\,\vec{r}_i \wedge \vec{v}_i$. For an extended solid, with moment of inertia I with respect to an axis which contains O, then $\vec{L} = I\vec{\omega}$, where $\vec{\omega}$ is the angular velocity of the body. If we express the velocity of an arbitrary particle $\vec{v}_i = \vec{\omega} \wedge \vec{r}_i$, we can rewrite the angular momentum as follows:

$$\vec{L} = \sum_{i=1}^{N} m_i\,\vec{r}_i \wedge (\vec{\omega} \wedge \vec{r}_i) = \sum_{i=1}^{N} m_i\,(\vec{r}_i \cdot \vec{r}_i)\vec{\omega}\ - (\vec{r}_i \cdot \vec{\omega}\ )\vec{r}_i \tag{2.1}$$

Alternatively, for extended systems:

$$\vec{L} = \int_M \vec{r} \wedge (\vec{\omega} \wedge \vec{r})dm = \int_M ((\vec{r} \cdot \vec{r})\vec{\omega}\ - (\vec{r} \cdot \vec{\omega}\ )\vec{r})dm \tag{2.2}$$

After decomposing the former expression in Cartesian components, and writing the resulting system in matrix form, we get:

$$\begin{bmatrix} L_\mathrm{x} \\ L_\mathrm{y} \\ L_\mathrm{z} \end{bmatrix} = \underbrace{\sum_{i=1}^{N} m_i \begin{bmatrix} (y_i^2 + z_i^2) & -x_i\,y_i & -x_i z_i \\ -x_i y_i & (x_i^2 + z_i^2) & -y_i z_i \\ -x_i z_i & -y_i z_i & (x_i^2 + y_i^2) \end{bmatrix}}_{\{I\}} \begin{bmatrix} \omega_\mathrm{x} \\ \omega_\mathrm{y} \\ \omega_\mathrm{z} \end{bmatrix} \tag{2.3}$$

where I is the inertia tensor, whose automated measurement is the main goal of the present work. By analogy, for a continuous system:

$$m_i \longrightarrow dm$$
$$\sum_{i=1}^{N} \longrightarrow \int_M$$

Therefore, the moments of inertia (the diagonal terms of the matrix I) will be:

$$I_{\mathrm{xx}} = \sum_{i=1}^{N} m_i \left(y_i^2 + z_i^2\right) \Longrightarrow I_{\mathrm{xx}} = \int (y^2 + z^2)\, dm$$
$$I_{\mathrm{yy}} = \sum_{i=1}^{N} m_i \left(x_i^2 + z_i^2\right) \Longrightarrow I_{\mathrm{yy}} = \int (x^2 + z^2)\, dm$$
$$I_{\mathrm{zz}} = \sum_{i=1}^{N} m_i \left(x_i^2 + y_i^2\right) \Longrightarrow I_{\mathrm{zz}} = \int (x^2 + y^2)\, dm$$

And the products of inertia (the off-diagonal terms of the matrix I) will be:

$$I_{\mathrm{xy}} = -\sum_{i=1}^{N} m_i\, x_i y_i = I_{y,x} \Longrightarrow I_{\mathrm{xy}} = -\int xy\, dm = I_{\mathrm{yx}}$$
$$I_{\mathrm{xz}} = -\sum_{i=1}^{N} m_i\, x_i z_i = I_{\mathrm{zx}} \Longrightarrow I_{\mathrm{xz}} = -\int xz\, dm = I_{\mathrm{zx}}$$
$$I_{\mathrm{yz}} = -\sum_{i=1}^{N} m_i\, y_i z_i = I_{\mathrm{zy}} \Longrightarrow I_{\mathrm{yz}} = -\int yz\, dm = I_{\mathrm{zy}}$$

As it is defined, the inertia tensor is given for a specific body in a specific frame of reference. Therefore, if we change the frame of reference, the inertia tensor of the body will accordingly change.

The inertia tensor has another very important property by construction: it is real and symmetric ($I_{\mathrm{ij}} = I_{\mathrm{ji}}$) and thus, it can always be diagonalized. Diagonalization leads to 3 *eigenvalues* and their corresponding *eigenvectors*. The physical concept behind the *eigenvectors* (in the present context) corresponds to a set of 3 right-handed orthogonal axes, that is, a new frame of reference for which the tensor of inertia of the object is diagonal. The eigenvalues would be the new moment of inertia in the new frame of reference. A set of right-handed orthogonal axes which makes the tensor of inertia of a body diagonal is referred to as set of *principal axes of inertia* for that body. The body will be able to rotate about any of its principal axes of inertia with constant angular velocity without the need of external torques.

2.1.2. Parallel axis theorem (Steiner's theorem)

The theorem in its 1-dimensional formulation proves that, given an axis which passes through the center of mass of a solid, and another axis parallel to the first one, the moment of inertia of both axes are related as as follows:

$$I_{\mathrm{P}} = I_{\mathrm{G}} + m d^2 \qquad \text{(1D)} \tag{2.4}$$

Where I_{G} Is the moment of inertia of the solid given for an axis going through the center of mass, I_{P} is the moment of inertia of the solid given for the parallel axis, m is the solid mass and d is the perpendicular distance between the two axes. Applying this to a 3D space, if we know the tensor of inertia tensor for an object with respect to a frame of reference centred in its centre of mass I_G, we can easily obtain the new tensor of inertia the object with respect to a new frame of reference I_P simply by calculating:

$$\left\{I_{\mathrm{P}}\right\} = \left\{I_{\mathrm{G}}\right\} + m \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -yx & x^2+z^2 & -yz \\ -zx & -zy & x^2+y^2 \end{bmatrix} \quad \text{(3D)} \tag{2.5}$$

Where m is the mass of the object, and x, y and z are the coordinates of the origin of the new frame of reference with respect to the one centred in the centre of mass. The parallel axis or Steiner theorem will be frequently used throughout this project.

2.1.3. Rotation using Euler angles formulation

Rotation matrices can be active (when they represent the actual rotation of a vector while the frame of reference is at rest), or passive (when the vector remains at rest and the frame of reference rotates). A passive rotation matrix is the inverse (or the transposed) of its equivalent active rotation matrix.

A rotation matrix of an angle θ about the X, Y and Z axes are, respectively:

$$R_{\mathrm{x}}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \quad R_{\mathrm{y}}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_{\mathrm{z}}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

We can apply a rotation matrix R on an arbitrary vector $\vec{v}$ as follows:

$$\vec{v}_{\mathrm{rotated}} = R\,\vec{v}$$

Matrix product allows us to apply consecutive rotations. For instance, if rotation R_1 is followed by R_2:

$$\vec{v}_{\mathrm{rotated}} = R_2\,R_1\,\vec{v}$$

Note that the fact that matrix products are not commutative is reflected on the fact that, when performing matrix rotations, the order matters.

Any arbitrary rotation can be decomposed in three consecutive rotations about different coordinate systems. For instance, if a passive rotation of an angle γ about an X axis is followed by a passive rotation of an angle β about an Y' axis, and by a passive rotation of an angle α about a Z axis, we will have:

$$R_{\mathrm{z}}(\alpha)R_{\mathrm{y}}(\beta)R_{\mathrm{x}}(\gamma) = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \tag{2.7}$$

Rotation matrices can be applied on other matrices and, specifically, can be applied to the inertia tensor $\left\{I\right\}$:

$$\left\{I_{\mathrm{rotated}}\right\} = R\,\left\{I\right\}\,R^T$$

2.1.4. Euler's equations

The fundamental equation of rotational dynamics in an inertial frame of reference relates the time variation of the angular momentum of a system ($\vec{L}$) with the external torques applied on that system ($\vec{\tau}$) as follows:

$$\frac{d\vec{L}}{dt} = \underbrace{\vec{\tau}}_{\text{sum of all external torques}} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{2.8}$$

The former expression given as above, that is, in an inertial frame of reference, is not efficient to deal with 3D rotation. The reason is that $\vec{L}_{\text{SRI}} = I_{\text{SRI}}\vec{\omega}_{\text{SRI}}$, and the actual expression of the inertia tensor I_{SRI} would vary (and should be recalculated) whenever the object moved with respect to the frame of reference. Therefore it is more convenient to express the inertia tensor in a frame of reference which allows its components to be constant, regardless the motion of the object. Such frame of reference must be be attached to the body (SRB) and, for simplicity, a frame of principal axes of inertia, so that I_{SRB} can be diagonal.

In order to relate the time variation of $\vec{L}$ in an inertial frame of reference in which Newton's Principles work (Equation 2.8), and the convenient frame of reference (which will be rotating and, thus, non-inertial), we can use the expression:

$$\vec{\tau} = \underbrace{\frac{d\vec{L}}{dt}}_{\text{Time variation in non-rotating SR, SRI}} = \underbrace{\frac{d^*\vec{L}^*}{dt}}_{\text{Time variation in rotating SR, SRB}} + \vec{\omega} \wedge \vec{L}^* \tag{2.9}$$

Expressing $\vec{L}^*$ in terms of the inertia tensor in the body frame: $\vec{L}^* = \{I\}\,\vec{\omega}$ because $\{I_{\text{SRB}}\}$ and working out some basic algebra, we can get Euler's equations for rotation, which are crucial to interpret rotation in 3D:

$$\begin{aligned} \tau_1 &= I_1\,\dot{\omega}_1 + (I_3 - I_2)\,\omega_3\,\omega_2 \\ \tau_2 &= I_2\,\dot{\omega}_2 + (I_1 - I_3)\,\omega_1\,\omega_2 \\ \tau_3 &= I_3\,\dot{\omega}_3 + (I_2 - I_1)\,\omega_2\,\omega_1 \end{aligned} \tag{2.10}$$

2.2. Pendulum movement description

The movement of the pendulum in question can be described as a result of the torque the torsion fiber generates. In addition of a restoring torque proportional to the rotated angle (-$b\,\theta$), real torsion fibers experience the effect of friction proportional to the time variation of the angle (-$b_f\dot{\theta}$). Therefore, if we apply that the sum all torques on an object with a extended moment of inertia I in the axis of the torsion fiber ($\sum \tau^{\text{ext}} = I\,\ddot{\theta}$), and both the τ and I are given with respect to the same point, the resulting equation of motion is:

$$-b\theta - b_f\,\dot{\theta} = I\,\ddot{\theta} \implies \ddot{\theta} + \frac{b_f}{I}\,\dot{\theta} + \frac{b}{I}\,\theta = 0$$

Introducing the expressions $\gamma = \frac{b_f}{2I}$ (the friction constant of the oscillator) and $\omega_0 = \sqrt{\frac{b}{I}}$ (its natural frequency), the equation above can be solved for he case $\gamma < \omega_0$. The resulting expression will be the equation for the motion of underdamped oscillators.

$$\theta = \theta_0 \, e^{-\gamma t} \cos(\omega t + \zeta) \tag{2.11}$$

Where: $\omega = \sqrt{\omega_0^2 - \gamma^2}$ is the actual angular frequency of the system, t is the time, θ_0 is the maximum amplitude of the oscillation, and ζ is its phase. The variables θ_0 and ζ are the determined by the initial conditions of each. The imposed condition $\gamma < \omega_0$ can be achieved by choosing a torsion fiber with suitable properties. This condition is set because, for our purposes, we are interested in a pendulum capable of enduring several oscillations before its amplitude decreasing significantly. An example of underdamped oscillator is shown in Figure 2.1.

In the expression 2.11 it is shown that the angular frequency of oscillation ω will be constant, and will only be dependant of the properties of the device. As a result, if the period of the oscillations ($T = \frac{\omega}{2\pi}$) is measured well enough, the system properties can be computed, in particular, the information leading to the inertia tensor. As we aim to measure the oscillation period and the amplitude of the oscillation will exponentially decrease with time ($\propto e^{-\gamma t}$), our interest will be in devices with relatively low γ

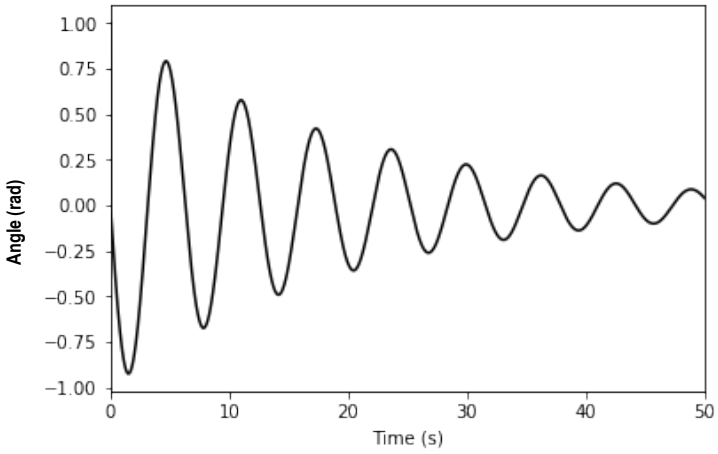


Figure 2.1: Example of an underdamped system graph.

CHAPTER 3. MAIN PROCEDURE

The overall aim of this chapter is to describe how to determine both the inertia tensor (I_{CM}) and the coordinates of the center of mass ($\vec{CM}$), with respect to the geometric centre of the satellite.

In order to provide a general description of the experimental setup our work is referred to, we first describe the main components of the torsion pendulum and their functions Then, we will present a step-by-step description of the procedure, along with depictions of the different configurations. Finally, we will complement our description with a brief review of all the steps taken.

3.1. The torsion pendulum

In this section the design of the chosen torsion pendulum will be reviewed as well as its different components and their functions. Firstly, we will inspect a figure showing a design outline and afterwards a *SolidWorks* design will be shown.

The pendulum's components can be divided into two main groups: The oscillatory parts and the fixed parts. The oscillatory parts are the parts which actually move with a pattern described in 2.2., and include the *CubeSat* platform, the support beam and the torsion fiber. The main non oscillatory parts are the components fixed to the ground, which support the pendulum and hold different measurement devices. Figure 3.1 shows the preliminary design of the pendulum with the non-oscillatory parts pictured in red.

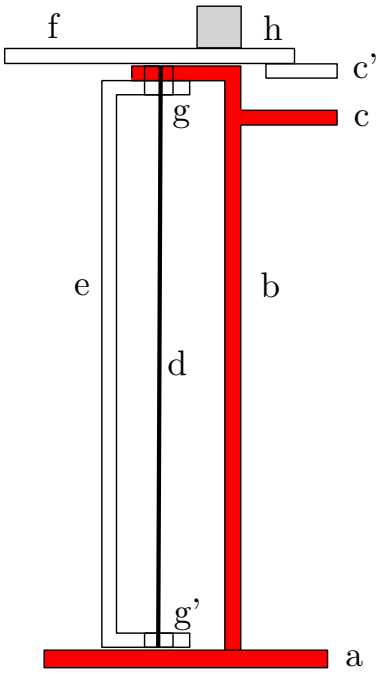


Figure 3.1: Figure of pendulum design with fixed parts in red and *CubeSat* in grey. Labels associated to each component are shown in tables 3.2 and 3.1. Credit (1).

Tables 3.1 and 3.2 show, respectively, the names of the oscillatory (white) and fixed (red) parts depicted in Figure 3.1.

Label in 3.1	Component name
c'	Laser arm
d	Torsion Fiber
e	Oscillatory support column
f	Oscillatory *CubeSat* platform
g and g'	bearings
h	*CubeSat*
Not depicted	Angled Support

Table 3.1: Names of oscillatory components of 3.1 (White in Figure 3.1).

Name in 3.1	Part name
a	Supporting Base
b	Fixed support column
c	Laser fixed arm

Table 3.2: Names of fixed components of 3.1 (Red in Figure 3.1).

The main function of the fixed parts (red) are to hold the structure and to allow the mobile parts to oscillate as freely as possible without interfering in the motion. Two bearings (c and c' in Figure 3.1) connect the fixed parts with the oscillatory ones, affecting the movement as little as possible while providing a firm support. The torsion fiber, labelled d in Figure 3.1, is connected to the oscillatory components at its upper end, and to the fixed platform at its lower end. Its torsion allows the actual oscillation. One of the most important parts of the design is the oscillatory platform, f in Figure 3.1, whose task is to provide a firm and precise way to attach the *CubeSat*, while simultaneously provide a number of locations to place it. For this purpose, in its surface there is a 9×9 mesh of holes at which the *CubeSat* can be screwed. A support to hold the *CubeSat* on its vertex (not depicted) can also be attached, should the need arise.

The period of the pendulum will be measured by a laser attached to the arm c which points at arm c'. Mounted in the oscillatory arm there will be a a photocell to measure the oscillation periods.

3.2. Procedure Steps and Math

The pendulum described above allows for the determination of its own moment of inertia along the vertical axis by taking measurements of oscillation periods with no test objects on the upper plate . Consequently, the measured moment when a satellite is on the plate will be the addition of both the moment of the object placed on the pendulum, and the one of the oscillating section itself. These measurements and the moments of inertia are related as follows:

$$I = \left(\frac{T}{2\pi}\right)^2 b \tag{3.1}$$

This expression is deduced from the original equation that describes the pendulum moment 2.11. As seen in 3.1 the inertia tensor of the whole system can be extracted knowing only the period T and the torsion constant b of the pendulum.

Throughout the whole experiment each period determination will be the mean of a number of separate experimental measurements. This will hep us mitigate the non-systematic or random errors in the measurements. Furthermore, the inertia tensor of the bare pendulum (3.2.1.) will be subtracted in order to extract the component of the object placed on it.

Overall, the procedure will be divided into two main blocks. In block one (*MomIn*), the inertia moments ($I_{\mathrm{CG,xx}}, I_{\mathrm{CG,yy}}, I_{\mathrm{CG,zz}}$), and the coordinates of the center of mass (X_0, Y_0, Z_0) will be obtained. Whereas in block two (*ProdIn*) the goal will be to obtain the inertia products ($I_{\mathrm{CG,xy}}, I_{\mathrm{CG,xz}}, I_{\mathrm{CG,yz}}$). As described in the *CubeSat* standard, we will use the geometric center of the cube as the origin of our coordinate system. However, the method we propose allows to measure the inertia tensor with respect to any chosen origin if the should the need arise. Specifically, the method will also yield the tensor with respect to the center of mass I_{CM}.

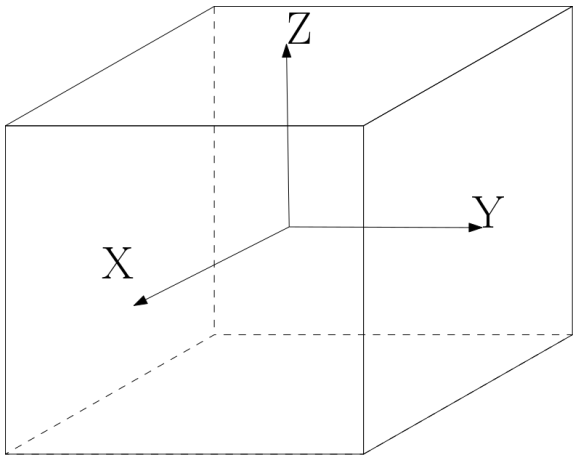


Figure 3.2: Representation of the Cubesat and the chosen coordinate system

3.2.1. Calibration

Before we begin experimenting with the *CubeSat*, a calibration process must be followed. For this procedure, we will perform the oscillation experiment with no *CubeSat* attached to the pendulum. In order to reduce the error, we will measure the period in a number of separate instances and use their average. With this period in hand, Equation 3.1 will be used. This will yield the moment of inertia of the pendulum about its rotating axis, seen in the equations as I_{Cal}. In all the experiments, in order to know the component of whatever is placed on the pendulum this I_{Cal} must be subtracted.

3.2.2. Block 1: Principal moments of inertia

Block (*MomIn*) will be responsible for calculating the inertia moments of the tensor (that is, its diagonal elements) with respect to its center of mass as, well as with respect to its geometric centre. Additionally, the coordinates to the center of mass will be extracted (X_0, Y_0 and Z_0).

3.2.2.1. *Sub-block 1 zz:* $I_{CG,zz}$

In the first step, we will calculate the $I_{CG,zz}$ component of the tensor. For this purpose, we will place the *CubeSat* on the platform with its Z axis pointing upwards. When measuring the $I_{CG,zz}$ component, bear in mind that the coordinates of the tensor of inertia (X_0, Y_0) will also be unknown, and that we need to take them into account.

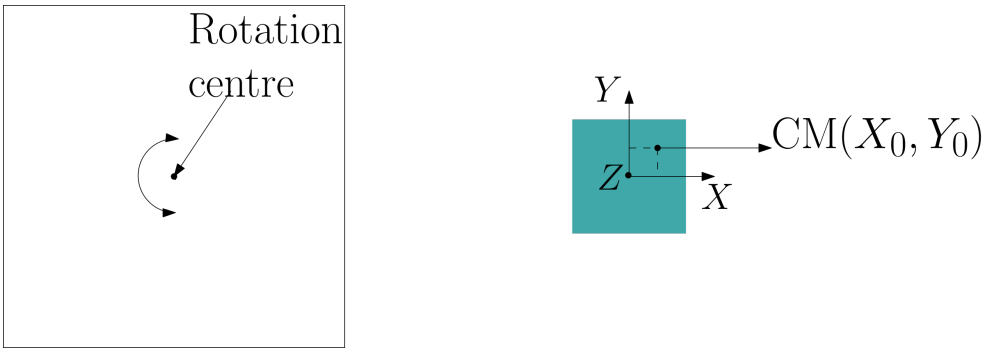


Figure 3.3: Preliminary Setup: Platform on the right *CubeSat* on the left(Viewed from above)

Firstly, we will place the *CubeSat* on the platform and displace it on the direction of the Y axis, as seen in Figure 3.4. It is critical that this displacement is made strictly along this direction. That is, that the rotating centre of the pendulum is contained within the Y axis of the *CubeSat*.

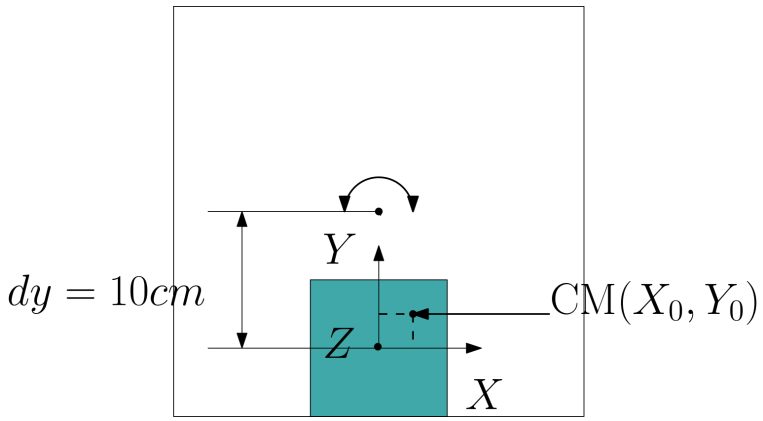


Figure 3.4: Illustration of the first experiment

This experiment will yield a tensor I_{1zz} that is equal to the searched $I_{CG,zz}$ displaced from the center of rotation. The expression of this tensor can be obtained using Steiner's theorem (2.1.2.) and, as a result, we get the following equation:

$$I_{1zz} = I_{CM,zz} + m[(dy - Y_0)^2 + X_0^2] \tag{3.2}$$

For our purposes, the displacement dy will be 10 cm but it can be changed as deemed necessary.

Secondly, we will place the *CubeSat* displaced along the X axis, as illustrated in Figure 3.5. Again, it is mandatory to take a special care in making sure that the point of rotation is contained within the X axis.

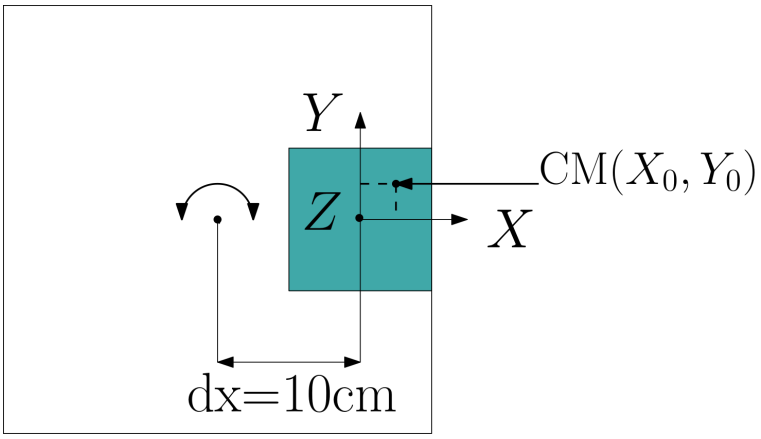


Figure 3.5: Illustration of the second experimental step

As a result, this experiment will give another tensor component I_{2zz}. We can follow the same logic as when using I_{1zz} but this time changing displacement in Steiner's theorem:

$$I_{2zz} = I_{CM,zz} + m[(dx + X_0)^2 + Y_0^2] \tag{3.3}$$

being dx a displacement of 10 cm too.

Finally, we will repeat the experiment but, in this case, we will make sure that the geometric center of the cube is contained in the axis of rotation of the pendulum. In this way, we obtain I_{3zz} which will provide us the $I_{CM,zz}$ component.

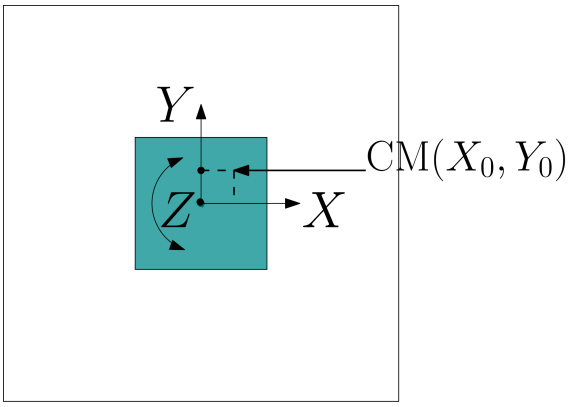


Figure 3.6: Illustration of the third experiment

We will obtain the final moment with the equation:

$$I_{3zz} = I_{CM,zz} + m(X_0^2 + Y_0^2) = I_{CG,zz} \tag{3.4}$$

After following these steps, three separate equations (3.2,3.3 and 3.5) with three unknowns(X_0, Y_0 and $I_{CG,zz}$); will be obtained. Evidently, an equation system can be formed and all three unknowns can be isolated.

$$\begin{cases} I_{1zz} = I_{CM,zz} + m[(dy - Y_0)^2 + X_0^2] \Longrightarrow I_{1zz} = I_{3zz} + m(dy^2 - 2Y_0 dy) \\ I_{2zz} = I_{CM,zz} + m[(dx + X_0)^2 + Y_0^2] \Longrightarrow I_{2zz} = I_{3zz} + m(dx^2 + 2X_0 dx) \\ I_{3zz} = I_{CM,zz} + m(X_0^2 + Y_0^2) = I_{CG,zz} \end{cases} \tag{3.5}$$

The component I_{3zz} measured in 3.5 is also the moment of inertia with respect to the geometric centre $I_{CG,zz}$, given that the *CubeSat* spins around its centre, so we can already say that $I_{CG,zz} = I_{3zz}$.

The coordinates X_0 and Y_0 can be solved for in Equation 3.5:

$$Y_0 = -\frac{I_{1zz} - I_{3zz}}{2mdy} + \frac{dy}{2} \tag{3.6}$$

$$X_0 = \frac{I_{2zz} - I_{3zz}}{2mdx} - \frac{dx}{2} \tag{3.7}$$

Once we have the coordinates, we can substitute them in the equation 3.5 in order to obtain $I_{CM,zz}$:

$$I_{CM,zz} = I_{3zz} - m(X_0^2 + Y_0^2) \tag{3.8}$$

As a result we have obtained values for all three unknowns ($I_{CM,zz}, X_0$ and Y_0). It is important to remember that for each of the values of I_{1zz}, I_{2zz} and I_{3zz}; a number of separate experiments have been carried out and the mean of all these periods has been put through the equation 3.1. So, if we were to measure each period 4 times, a total of twelve experiments would have been carried out for this sub-block.

For the calculus of the remaining two inertia moments ($I_{CM,yy}$ and $I_{CM,xx}$), the method will be very similar but with the *CubeSat* appropriately rotated in such a way that, the rotating axis, and our desired axis are parallel.

3.2.2.2. Sub-block 2 yy: $I_{CG,yy}$

In order to measure this component the *CubeSat* will be rotated 90° around its X axis yielding the ~~following~~ setup shown in Figure 3.7. To make sure that the measured component is the desired one, we can use the rotation matrix (2.1.3.) with a rotation angle $\gamma = 90°$, and obtain that, coincidentally, $I_{Rotated,zz} = I_{yy} = I_{Measured}$. This is evident when looking at the setup from a spatial point of view, seeing as the *CubeSat* will oscillate only around its Y axis

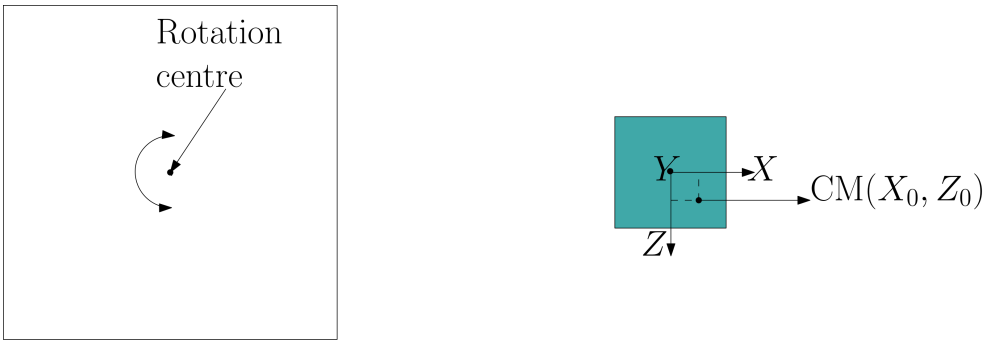


Figure 3.7: Preliminary Setup: Platform on the right *CubeSat* on the left(Viewed from above)

Following the same procedure as in Figure 3.4 for the first experiment, the *CubeSat* will be displaced in its vertical axis as seen in Figure 3.8 yielding the equation 3.9.

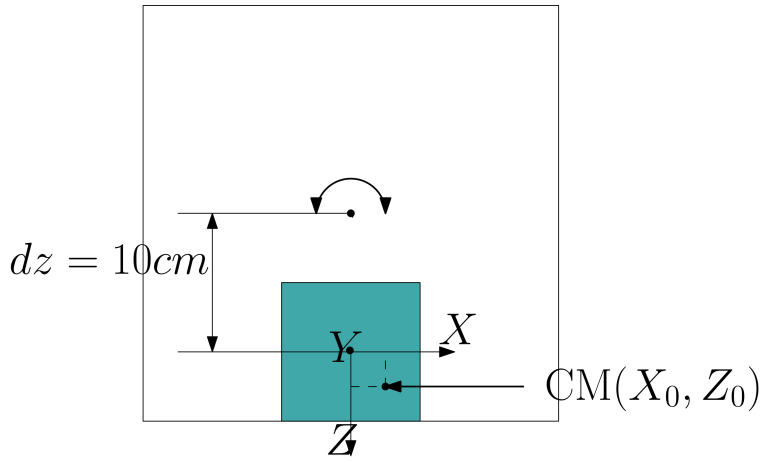


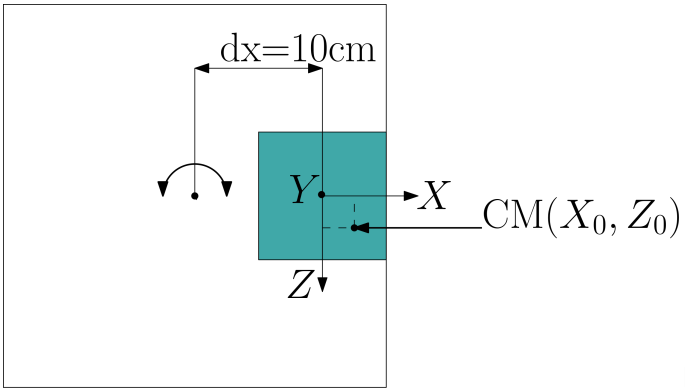
Figure 3.8: Illustration of the first experiment

That gives us:

$$I_{1yy} = I_{CM,yy} + m[(dz + Z_0)^2 + X_0^2] \tag{3.9}$$

As before we have chosen the magnitude dz to be of 10 cm but it can be changed as required.

For the second experiment we, again, displace the *CubeSat*, this time along the X axis.



b

Figure 3.9: Illustration of the second experiment

As result we obtain the following:

$$I_{2yy} = I_{CM,yy} + m[(dx + X_0)^2 + Z_0^2] \tag{3.10}$$

being dx a displacement of 10 cm too.

The final experiment will again consist of placing the *CubeSat* with its geometric centre on the rotating axis.

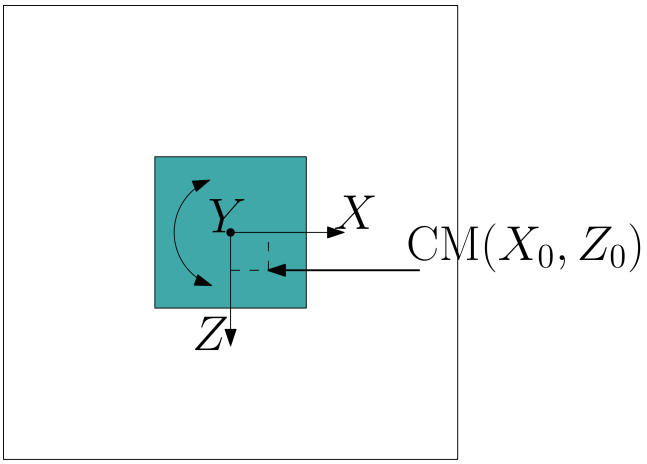


Figure 3.10: Illustration of the third experiment

With the equation being:

$$I_{3yy} = I_{CM,yy} + m(X_0^2 + Z_0^2) = I_{CG,yy} \tag{3.11}$$

In this experiment we also find the $I_{CG,yy}$ component directly as a result of our *CubeSat* rotating around its geometric centre. The equation system will this time be:

$$\begin{cases} I_{1yy} = I_{CM,yy} + m[(dz + Z_0)^2 + X_0^2] \Longrightarrow I_{1yy} = I_{3yy} + m(dz^2 + 2Z_0 dz) \\ I_{2yy} = I_{CM,yy} + m[(dx + X_0)^2 + Z_0^2] \Longrightarrow I_{2yy} = I_{3yy} + m(dx^2 + 2X_0 dx) \\ I_{3yy} = I_{CM,yy} + m(X_0^2 + Z_0^2) = I_{CG,yy} \end{cases} \tag{3.12}$$

Operating with the equations in 3.12, we obtain:

$$Z_0 = \frac{I_{1yy} - I_{3yy}}{2mdz} - \frac{dz}{2} \tag{3.13}$$

$$X_0 = \frac{I_{2yy} - I_{3yy}}{2mdx} - \frac{dx}{2} \tag{3.14}$$

Substituting these variables in the equation 3.11:

$$I_{CM,yy} = I_{3yy} - m(X_0^2 + Z_0^2) \tag{3.15}$$

With this, we have obtained, on one hand, the $I_{CM,yy}$ and the $I_{CG,yy}$ components, and, on the other hand, two coordinates for the center of mass. Theoretically, the calculus of X_0 is redundant in this case, as it has been previously computed. Nevertheless, this method over-defines the three coordinates of the center of mass as a means to reduce the error. The final coordinates will be the average of the two values obtained for each X_0, Y_0 and Z_0.

3.2.2.3. Sub-block 3 xx: $I_{CG,xx}$

In this final sub-block of Block (*MomIn*) the desired component of the tensor is the $I_{CM,xx}$. The easiest way to compute it is to make sure that the axis X of the *CubeSat* is parallel to the rotating axis of the platform. In order to do this, the *CubeSat* will be rotated -90° around its Y axis which gives us the following configuration.

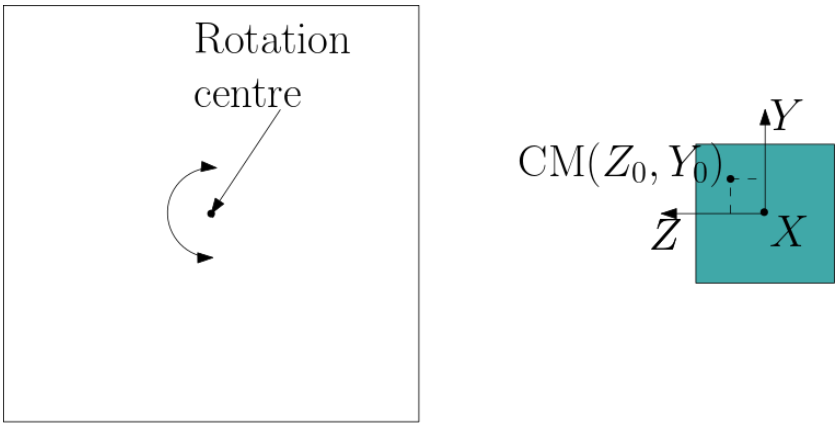


Figure 3.11: Preliminary Setup: Platform on the right, *CubeSat* on the left (viewed from above)

As usual, for the first experiment the *CubeSat* will be placed with an offset dy along its Y axis:

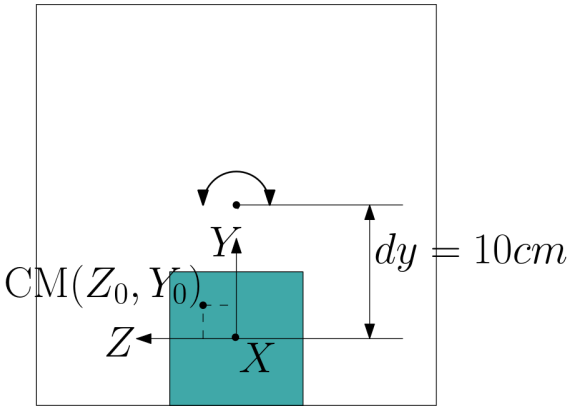


Figure 3.12: Illustration of the first experiment

Therefore:

$$I_{1\mathrm{xx}} = I_{\mathrm{CM,xx}} + m[(dy - Y_0)^2 + Z_0^2)] \tag{3.16}$$

Following our standard procedure, we have also chosen dy to be of 10 cm.

In the second experiment, the *CubeSat* will be displaced in the direction given by its Z axis.

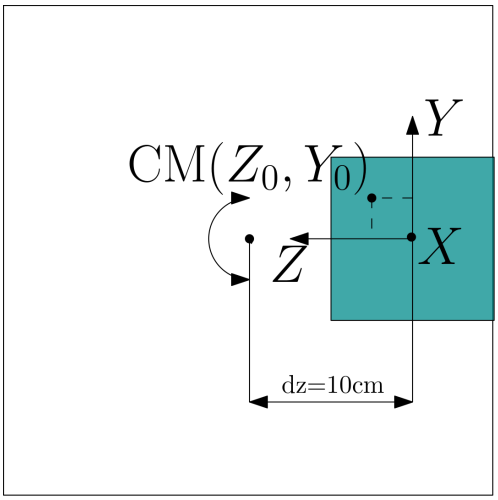


Figure 3.13: Illustration of the second experiment

Resulting in the following:

$$I_{2xx} = I_{CM,xx} + m[(dz - Z_0)^2 + Y_0^2] \tag{3.17}$$

being again dz a displacement of 10 cm.

This block will be finished by placing the *CubeSat* on the rotating axis of the platform.

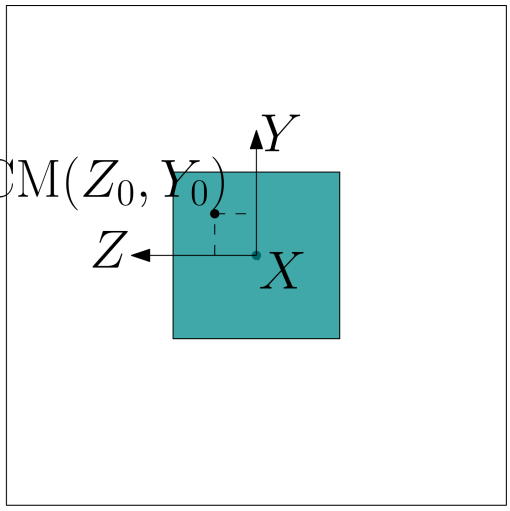


Figure 3.14: Illustration of the third experiment

Hence, the equation will be:

$$I_{3xx} = I_{CM,xx} + m(Y_0^2 + Z_0^2) = I_{CG,xx} \tag{3.18}$$

We can obtain $I_{CG,xx}$ directly from this experiment.

The final equation system will be:

$$\begin{cases} I_{1xx} = I_{CM,xx} + m[(dy + Y_0)^2 + Z_0^2] \Longrightarrow I_{1xx} = I_{3xx} + m(dy^2 + 2Y_0 dy) \\ I_{2xx} = I_{CM,xx} + m[(dz - Z_0)^2 + Y_0^2] \Longrightarrow I_{2xx} = I_{3xx} + m(dz^2 + 2Z_0 dz) \\ I_{3xx} = I_{CM,xx} + m(Z_0^2 + Y_0^2) = I_{CG,xx} \end{cases} \tag{3.19}$$

Operating with the equations 3.19:

$$Y_0 = \frac{I_{1xx} - I_{3xx}}{2mdy} - \frac{dy}{2} \tag{3.20}$$

$$Z_0 = \frac{I_{2xx} - I_{3xx}}{2mdz} - \frac{dz}{2} \tag{3.21}$$

Substituting these variables in the equation 3.11:

$$I_{\mathrm{CM,yy}} = I_{3yy} - m(X_0^2 + Z_0^2) \tag{3.22}$$

This ends our first block of experiments.

*3.2.2.4. Conclusions and Summary of Block (*MomIn*)*

In this block, the three inertia moments of our *CubeSat* have been computed for a system of reference fixed on the geometric centre of the *CubeSat* ($I_{\mathrm{CG,xx}}, I_{\mathrm{CG,yy}}$ and $I_{\mathrm{CG,zz}}$) as well as with respect to the center of mass ($I_{\mathrm{CM,xx}}, I_{\mathrm{CM,yy}}$ and $I_{\mathrm{CM,zz}}$). From the tensor point of view the moments of inertia, laying in the diagonal of the tensor, have been obtained, thus leaving the rest (the inertia products) to be calculated. Fortunately, the inertia tensor is symmetric, which means that we have to obtain only three additional components in order to complete our experiment.

Additionally, we have extracted all three coordinates of the center of mass with respect to the coordinate origin (X_0, Y_0, Z_0). Two separate configurations have been used to extract the values of each one of these coordinates. Theoretically, these values should be the same but, in practice, they can vary so, moving forward, we will use the mean of the two determinations as the final value.

As a last observation, the period of the pendulum has been measured a number of times for each single tensor component. This means that, if we measured each separate period 4 times, thus far, we would have measured it 4 times for calibration purposes, and $4 \times 3 \times 3$ to determine the inertia moments, we have already measured 40 separate periods.

3.2.3. Block 2: Inertia products (*ProdIn*)

In this block, the goal will be to compute the remaining components of the inertia tensor. For all these experiments, the *CubeSat* will need to be rotated by a certain angle and placed on the pendulum. For this purpose, the pendulum will be fitted with a special support that maintains the *CubeSat* in the desired position. As a result, a second calibration will be required in order to find the inertia tensor of the supporting device without a *CubeSat* on it(3.15).

One of the main problems derived from the need to rotate the *CubeSat* is that the distance between the rotation axis and the chosen frame of reference changes by a certain amount, resulting in a different tensor that must be determined by using Steiner theorem (2.1.2.). The easiest solution for this issue is to make sure that our origin for the inertia tensor is contained within the rotation axis, avoiding Steiner effects all together. Bear in mind that we have calculated the main moments with respect to two points: the center of mass (I_{CM}) and the geometrical center (I_{CG}). The *CubeSat* could be placed with either of these

points contained within the rotation axis, but it is much easier to use the geometrical centre because it is already predetermined and much more reliable.

Taking all of the above into account, the strategy in this block will be to find all of the remaining components of I_{CG} and, then, work backwards from that to find the I_{CM} making use of the center of mass coordinates(X_0, Y_0 and Z_0) and Steiner's theorem.

3.2.3.1. Calibration with support

The remaining experiments require to rotate the satellite a certain angle. We have chosen this angle to be of 45° for tha sake of simplicity. In order to achieve this, a support is placed on the pendulum thus changing its moment of inertia. A new calibration will need to be performed in order to take this change into account. The value extracted from this experiment(I_{Cal1}) will then be subtracted from all of the following ones, so that the moment of inertia of the *CubeSat* placed on the support can be found.

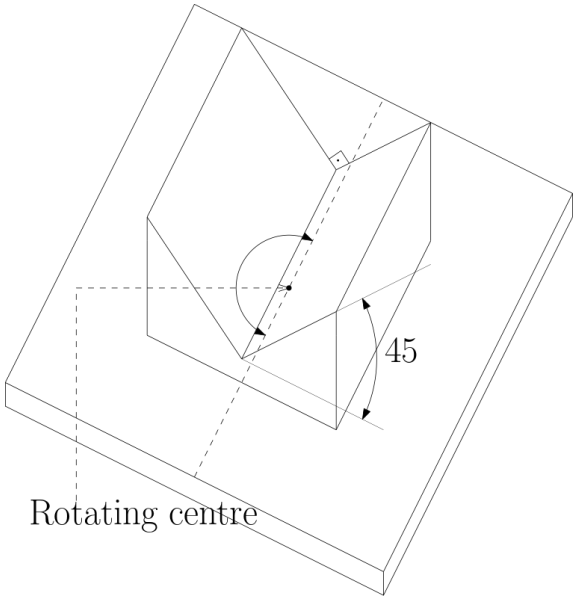


Figure 3.15: Illustration of the pendulum platform with the support attached to it.

3.2.3.2. Sub-block 1 yz: $I_{\mathrm{CG,yz}}$

In this sub-block the aim is to calculate $I_{\mathrm{CG,yz}}$. The easiest way to achieve this is to rotate the *CubeSat* 45° with respect to its X axis ($(\alpha, \beta, \gamma) = (0, 0, 45^\circ)$). An illustration of the setup can be seen in Figure 3.16, where we can see that the geometric center is contained within the rotation axis.

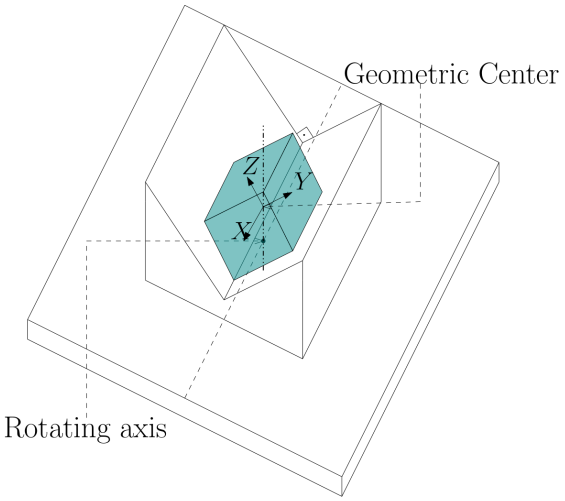


Figure 3.16: Illustration of the pendulum with the support and the *CubeSat* in blue. Notice the arrows for our coordinate system showing where the satellite is pointing.

With this attitude, we can use the rotation matrix (2.1.3.) to find the new moment (I_{Rot1}) and, particularly its component $I_{\text{Rot1,zz}}$, which is what we are measuring, as a function of variables we already know.

$$R = Rot(0,0,45^\circ) \tag{3.23}$$

$$I_{\text{Rot1}} = R\, I_{\text{CG}}\, R^T \longrightarrow I_{\text{Rot1,zz}} = \frac{1}{2}(I_{\text{CG,xx}} - 2I_{\text{CG,yz}} + I_{\text{CG,zz}}) \tag{3.24}$$

The variables $I_{\text{CG,xx}}$ and $I_{\text{CG,zz}}$ are already known from the previous block (3.19 and **??**), and we have just found $I_{\text{Rot1,zz}}$; as a result, we can isolate the unknown and compute its value:

$$I_{\text{CG,yz}} = \frac{I_{\text{CG,xx}} + I_{\text{CG,zz}} - 2I_{\text{Rot1,zz}}}{2} \tag{3.25}$$

3.2.3.3. Sub-block 2 xz: $I_{\text{CG,xz}}$

In this sub-block we will use a very similar method as in the previous one although this time, the *CubeSat* will be rotated 45° around its Y axis ($(\alpha,\beta,\gamma) = (0,45^\circ,0)$). The geometrical center is again contained within the rotating axis so, once more, the rotation matrix (2.1.3.) can be used to find the new tensor (I_{Rot2}) and, particularly its $I_{\text{Rot2,zz}}$ component

$$R = Rot(0,45^\circ,0) \tag{3.26}$$

$$I_{\text{Rot2}} = R\, I_{\text{CG}}\, R^T \longrightarrow I_{\text{Rot2,zz}} = \frac{1}{2}(I_{\text{CG,xx}} + 2I_{\text{CG,xz}} + I_{\text{CG,zz}} \tag{3.27}$$

Isolating for the variables we already know;

$$I_{\text{CG,xz}} = \frac{2I_{\text{Rot2,zz}} - I_{\text{CG,xx}} - I_{\text{CG,zz}}}{2} \tag{3.28}$$

3.2.3.4. Sub-block 3 xy: $I_{\text{CG,xy}}$

This final sub-block requires us to rotate the *CubeSat* twice. First 45° with respect to the Y axis and then, 90° around the X axis($(\alpha,\beta,\gamma) = (0,45^\circ,90^\circ)$), which is equivalent to taking the satellite from the previous step and rotating it 90° around is X axis.

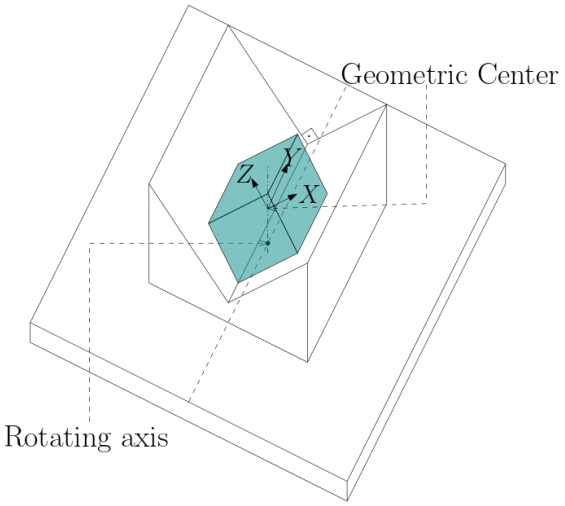


Figure 3.17: Illustration of the pendulum with the support and the *CubeSat* in blue. Notice the arrows for our coordinate system showing where the satellite is pointing.

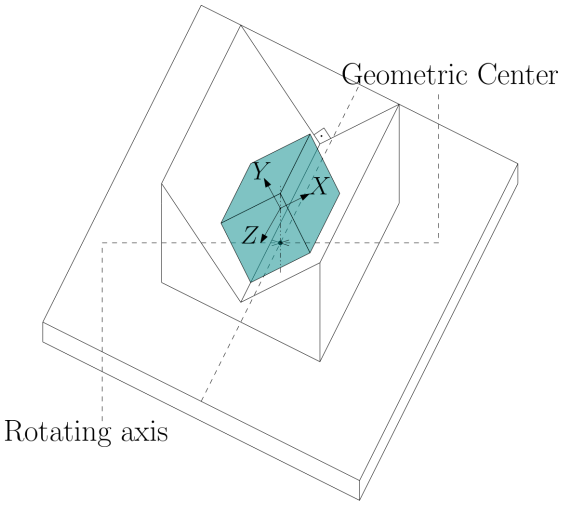


Figure 3.18: Illustration of the pendulum with the support and the*CubeSat* in blue. Notice the arrows for our coordinate system showing where the satellite is pointing.

As usual, the rotation matrix (2.1.3.) can be used to find the new tensor (I_{Rot3}) and, particularly its $I_{\text{Rot3,zz}}$ component:

$$R = Rot(0, 45°, 90°) \tag{3.29}$$

$$I_{\text{Rot2}} = R\, I_{\text{CG}}\, R^T \longrightarrow I_{\text{Rot2,zz}} = \frac{1}{2}(I_{\text{CG,xx}} + 2I_{\text{CG,xy}} + I_{\text{CG,zz}} \tag{3.30}$$

Isolating for the variables we already know;

$$I_{\text{CG,xz}} = \frac{2I_{\text{Rot3,zz}} - I_{\text{CG,xx}} - I_{\text{CG,yy}}}{2} \tag{3.31}$$

3.2.3.5. Complete I_{CM}

At this point of the experiment, we have already fully established I_{CG} and the diagonal for the I_{CM}. In addition, ~~to the fact that~~ we already know the coordinates for the center

of mass(X_0, Y_0, Z_0). Another interpretation of these coordinates is to view them as the position of the center of mass with respect to the geometrical center. Evidently Steiner's theorem (2.1.2.) can be used to relate both tensors and complete our experiment.

$$I_{\mathrm{CM}} = I_{\mathrm{CG}} - Steiner = \begin{bmatrix} I_{\mathrm{CG,xx}} & I_{\mathrm{CG,xy}} & I_{\mathrm{CG,xz}} \\ I_{\mathrm{CG,yx}} & I_{\mathrm{CG,yy}} & I_{\mathrm{CG,yz}} \\ I_{\mathrm{CG,zx}} & I_{\mathrm{CG,zy}} & I_{\mathrm{CG,zz}} \end{bmatrix} - m \cdot \begin{bmatrix} Y_0^2 + Z_0^2 & -X_0 Y_0 & -X_0 Z_0 \\ -Y_0 X_0 & X_0^2 + Z_0^2 & -Y_0 Z_0 \\ -Z_0 X_0 & -Z_0 Y_0 & X_0^2 + Y_0^2 \end{bmatrix} \tag{3.32}$$

3.3. Table with hierarchy of experiments

In this section the different blocks and sub-blocks of the experiment will be shown in table form along with the unknowns each step clears. To be thorough, the variables to compute are:

- I_{CM}= Inertia tensor with respect to the center of mass.
- I_{CG}= Inertia tensor with respect to the geometric centre.
- X_0, Y_0 and Z_0=Coordinates of the centre of mass.

Keep note that the coordinates for the centre of mass have been calculated twice each one so there will be some overlap.

Block	Sub-block	Subsection	Variable
1 (*MomIn*)	Calibration	1 (3.2.1.)	I_{Cal}
	1 (3.2.2.1.)	1 (3.2), 2 (3.3), 3 (3.5)	$I_{\mathrm{CG,zz}}, I_{\mathrm{CM,zz}}, X_0, Y_0$
	2 (3.2.2.2.)	1(3.9), 2 (3.10), 3 (3.11)	$I_{\mathrm{CG,yy}}, I_{\mathrm{CM,yy}}, X_0, Z_0$
	3 (3.2.2.3.)	1 (3.16) ,2 (3.17) ,3 (3.18)	$I_{\mathrm{CG,xx}}, I_{\mathrm{CM,xx}}, Z_0, Y_0$
2 (*ProdIn*)	Calibration 2	1 (3.2.3.1.)	I_{Cal1}
	1 (3.16)	1	$I_{CG,yz}$
	1 (3.17)	1	$I_{\mathrm{CG,xz}}$
	1 (3.18)	1	$I_{\mathrm{CG,xy}}$
Complete I_{CM} (3.32)		$I_{\mathrm{CM,xy}}, I_{\mathrm{CM,xz}}, I_{\mathrm{CM,zy}}$	

Table 3.3: Section explanation. Block 1 (*MomIn*) and Block 2 (*ProdIn*) refer, respectively, to the calculation of moments of inertia and of products of inertia respectively.

CHAPTER 4. PROGRAM

In this chapter we describe the inner working of the program with the aid of code fragments. Because of its sufficient power and its user-friendly configuration, we have chosen to program our code in MATLAB (r2019b). However, if another program language were deemed more convenient, the following description, which covers both the code philosophy and its details, should facilitate an eventual translation. Note that, along this section we will use the Block and sub-block nomenclature described in Table 3.3. The functions mentioned along the chapter are described at the end, in section 4.3.8.. A complete list of the used variables and a description of their function is available in appendix B.1.

4.1. Main aims of the program

The program we present has two main functions: in one hand, it can use the inputs provided by the user after performing a real-life experiment and obtain the centre of mass position and inertia tensor components. On the other hand, it can simulate the whole experiment from scratch and present the expected results.

In all of the cases the final results will be two 3×3 matrices corresponding to the inertia tensor with respect to the chosen origin of coordinates. In a one-unit *CubeSat* this origin usually corresponds to the geometric centre, called *CGtens* in the program, and the inertia tensor with respect to the centre of mass, called *CMtens*. Additionally, the coordinates of the centre of mass will be provided (Xcm, Ycm and Zcm). Once computed, these results will be presented to the user with the option to write them in a text file.

4.2. Main Structure

The overall program functions can be easily explained with the aid of a flow diagram (4.1). As it can be seen, the two main branches correspond to the two main functions of the program.

The first question that the users are asked is whether they would like to simulate an experiment (by introducing first information of a known object's mass, centre of mass position and inertia tensor), or introduce data from real measurements. Both of these options will eventually lead to the periods needed to perform the method of inertia tensor calculation previously explained in section 3.

4.2.1. Simulated input

If the users choose to perform a simulation of the experiment, the program asks them to introduce the tensor data of the object for which they want to perform the simulation. We allow three options for the users to introduce data:

- Manual input: The program reads the data from the main console (4.3.8.4.).
- File read: The program extracts the tensor data from a text file with a defined format (4.3.8.3.).

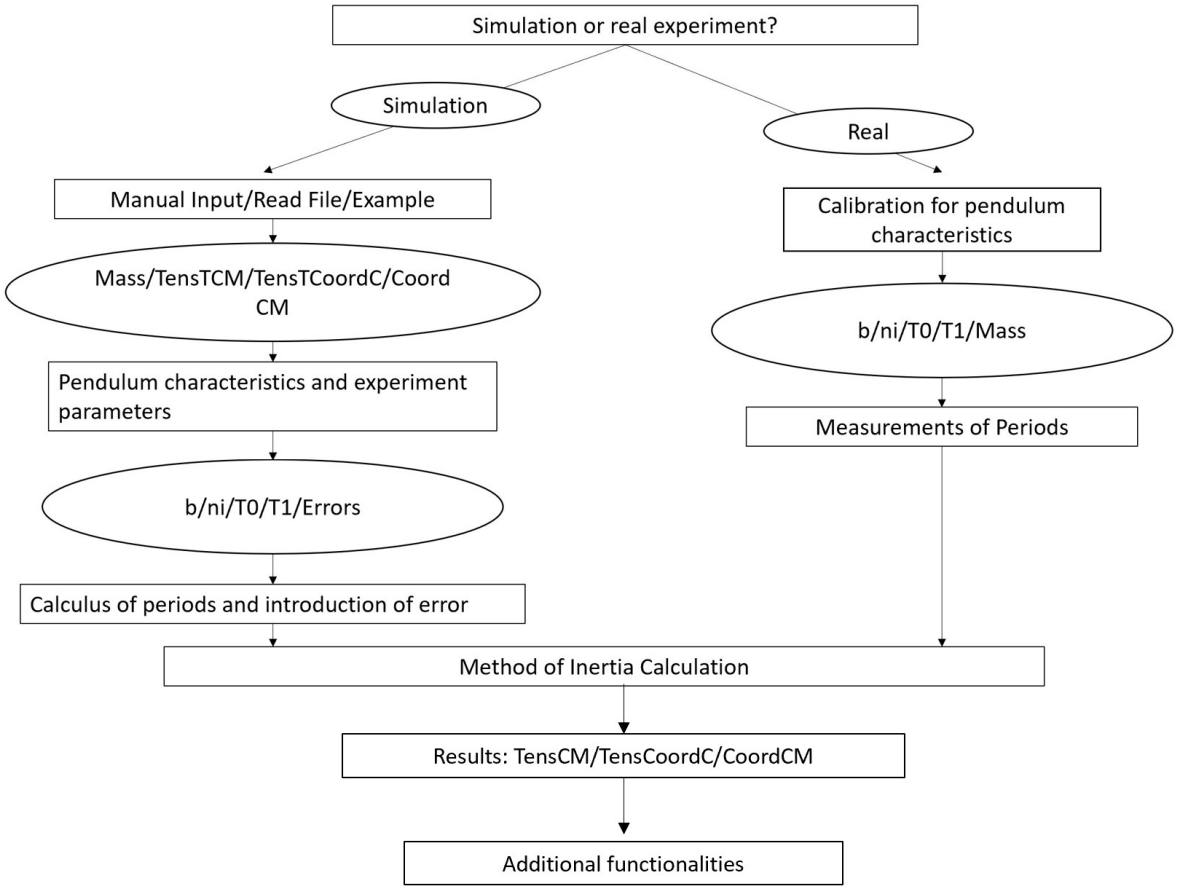


Figure 4.1: Algorithm for the program

- Example: The program uses a built-in function in order to analytically calculate the tensor data of a 10×10×10 cm cube with a displaced spherical hole in it (4.3.8.2.). This option is aimed to allow preliminary testing of the device's performance.

All these ways lead up to the same data set.

- m: The mass of the object (in our case a *CubeSat*) in kg.
- *TensTCM*: The 3×3 matrix of the inertia tensor of the object with respect to a frame of reference centred in its centre of mass in kg m^2.
- *TensTCG*: The 3×3 matrix of the inertia tensor of the object with respect to the chosen frame of reference (which must be parallel to the frame of reference centred in the centre of mass of the object) in kg m^2.
- *CoordCM*: The X, Y, Z coordinates of the centre of mass in m.

Once the data of the actual object is known, the next input the program needs is the characteristics of the equipment and the procedure:

- b= The torsion constant of the pendulum N·m^2.

- ni= The number of times each particular measurement is made.
- $I0$= The pendulum moment about an axis parallel to the torsion fiber with no *CubeSat* attached to it in kg m^2.
- $I1$= The pendulum moment with the 45° support attached in kg m^2.
- Additionally, the systematic and random error of different variables are also asked.

With all of this information the program can use the inverse of the formula 3.1, and derive the period of the pendulum with the object in the desired configuration.

$$T = (\sqrt{\frac{I}{b}})2\pi \tag{4.1}$$

Aiming to perform an accurate simulation, each time a period is needed to be computed, the program uses the equation above ni times, and each time, it introduces a random and systematic error whose width is asked to the user.
After this, the program has the necessary inputs (all the different periods in each configuration as if the experiment were performed in real life), to perform the procedure described in Chapter 3 and thus to calculate the inertia tensor.

4.2.2. Real Input

If the user chooses this option the program uses as the input all the data from a real experiment. Additionally, it gives the user instructions for completing the experiment, that is, to place the *CubeSat* in different specific configurations. Each time the period from a single configuration is asked, the program allows to enter as many period values as desired to reduce measurement errors.

The first information the program requires is the pendulum characteristics. These are extracted from the known variables of the pendulum and from the calibration process.

- b= The torsion constant of the pendulum in N·m^2. This variable can either be known, or measured using third party equipment.
- $T0$= The pendulum period with no *CubeSat* attached to it given in seconds. As in the simulated input case, the users can input as many period values as desired.
- $T1$= The pendulum period with the 45° support attached, given in seconds.

Once the characteristics of the equipment are known, the program guides the user in performing the experiment and asks for the required data as the measurements are performed.

4.2.3. Method of inertia tensor calculation

Once this point is reached the program has all of necessary period measurements to perform the inertia tensor calculation (see 3). This method results in three different output variables:

- *TensCM*: The 3×3 inertia tensor of the object with respect to a frame of reference centred in its centre of mass in kg m^2.
- *TensCG*: The 3×3 inertia tensor~~tensor~~ of the object with respect to the chosen frame of reference in kg m^2.
- *CoordV*: The X, Y, Z coordinates of the centre of mass in m.

These data are presented to the user in the main console, and the option is given to save them in a text file with the same format the program is capable of reading. In addition, in the event of choosing to simulate the experiment, a comparison is made between the extracted data and the original one.

4.3. Program description

In this section we will review the program code and explain the aim and operation of its different functionalities. The diverse functions used and their purpose will also be explained. Bear in mind that the code can be changed in order to accommodate other uses and, furthermore, it is encouraged to do so.

4.3.1. Main menu and program loop

As mentioned above, the first question the users are asked is if they want to either perform a simulation of the program or input data from a real experiment. This is done via a simple input of 0 for the input of real data or a 1 for performing a simulation. The program is written in such a way that, independently of the options the user chooses, the variables in order to perform the method explained in 3, will all be known by the time that step is reached. When the end of the program is reached, the user will always have the option to save the results and repeat the script.

Listing 4.1: Code asking the user the two options

```
1 while (repeat==1)
2     sim=2;
3     while ((sim≠0)&&(sim≠1))||(¬isfloat(sim))
4         sim=input('Would you like to simulate the experiment(1) ...
              or to input the data from a real measure(0): ');
```

As seen in the code fragment above (listing 4.1) the program will continuously ask for the required input until a valid value is written. Once the choice is made further experiment options are asked.

4.3.2. Input of real variables

If the user has chosen to input the data from a real experiment, the program will act both as a storage of that data and a guide, showing step by step instructions to the user in order

to perform the experiment as expected. This procedure is divided into sections that can be repeated should the need arise.

4.3.2.1. Calibration and experiment data

In this section the program asks the user for data regarding the setup and conditions of the experiment (see Listing 4.2).

Listing 4.2: Code asking the user for the experiment conditions

```
%The values of the calibration
repeat=0;
while (repeat==0)
    b=input('Please enter the value of b[N m^2]: \n');
    fprintf('Please measure the period of the pendulum ...
        without anything placed on it \n');
    fprintf('Make several measurements in order to ...
        lessen the effect of the random error(We recomend ...
        4) \n')
    T0=AskPeriod('Write the periods of the ...
        calibration[S]: \n','Input a period, if done ...
        input 0: \n');
    m=input('Please write the mass of the satellite[Kg] ...
        \n');
```

The stored variables are:

- b= The torsion constant of the pendulum in N·m^2. This variable can be known or measured using third party equipment.
- $T0$= The pendulum period with no *CubeSat* attached to it, in seconds. Users can input as many values of this period as needed (see Function 4.3.8.1.).
- m= The mass of the satellite, in kg.
- $repeat$= Variable that controls whether the section is repeated (1) or finished (any other value)

4.3.2.2. Input of Block 1: determination of moments of inertia

This section of the program (see Listing 4.3) asks the user for the data corresponding to the calculus of I_{zz} as seen in section 3.2.2.1..

Listing 4.3: Code asking for Block 1 Sub-block 1 of the experiment

```
end
repeat=0;
while (repeat==0)
%Ask the user for the values needed for Block 1
    fprintf('BLOCK 1: Principal moments \n ');
    %Measure of Izz
    fprintf('Measure of Izz(Experiment 1.1) \n ');
    fprintf('Please place the satellite on the platform ...
       with its Z axis pointing up \n' );
    fprintf('Move the satellite a certain distance from ...
       the rotation center making sure the rotation ...
       center is contained within the Y axis(Experiment ...
       1.1.1) \n');
    dy1=input('Input the displacement of the satellite ...
       in its vertical axis[m] \n');
    T1=AskPeriod('Write the periods of the experiment ...
       with the satellite displaced vertically [s] ...
       \n','Input a period, if done input 0: \n');
    fprintf('Move the satellite a certain distance from ...
       the rotation center making sure the rotation ...
       center is contained within the X axis(Experiment ...
       1.1.2) \n');
    dx1=input('Input the displacement of the satellite ...
       in its horizontal axis[m] \n');
    T2=AskPeriod('Write the periods of the experiment ...
       with the satellite displaced horizontally [s] ...
       \n','Input a period, if done input 0: \n');
```

The stored variables are:

- $T1, T2$ and $T3$= Vectors containing several period measurements for each subsection of the experiment, in seconds. The function $AskPeriod$ (4.3.8.1.) is used for this purpose. These vectors may not have the same length, which yields a challenge that will be further reviewed.
- $dy1$ and $dx1$= The distance the *CubeSat* has been moved in order to complete the subsections 3.4 and 3.5. This displacements must adhere to the sign convention of the chosen coordinate system so that the program may function properly.

In order to organize the considered periods, the associated vectors are stored in a matrix of 3×n dimensions called $Tmat1$. This yields a challenge because these vectors may not have the same length. In order to solve this, the dimension of the longest one is matched by filling the shorter ones with the mean of their value. This way we have 3 vectors with the same length and with an unaltered mean value. Bear in mind that this average value will be the one used for all the following calculations.

Here is a mathematical example of how the program would work in the event that the period vectors $\vec{T}1, \vec{T}2$ and $\vec{T}1$ have a length of 3,4 and 5 items; respectively.

$$Tmat1 = \begin{bmatrix} T11 & T12 & T13 & \overline{T1} & \overline{T1} \\ T21 & T22 & T23 & T24 & \overline{T2} \\ T31 & T32 & T33 & T34 & T35 \end{bmatrix} \tag{4.2}$$

The MATLAB code section in charge with this is as shown in Listing 4.4.

Listing 4.4: Code generating matrix $Tmat1$ with the input periods

```
columns=max([length(T1),length(T2),length(T3)]);
Tmat1=[T1,ones(1,(columns-length(T1)))*mean(T1);
    T2,ones(1,(columns-length(T2)))*mean(T2);
    T3,ones(1,(columns-length(T3)))*mean(T3)];
```

An analogous process is followed for all three sections of Block 1 (to determine I_{zz}, I_{yy} and I_{xx}), which generates the variables corresponding to the distances the *CubeSat* has been moved, and three matrices ($Tmat1, Tmat2$ and $Tmat3$) which include the periods. Each section can be repeated in the event the user has committed a mistake and wants to introduce the data again. As a summary, in this block we have the following variables.

- Sub-block 1: $dy1, dx1$ and $Tmat1$.
- Sub-block 2: $dz2, dx2$ and $Tmat2$.
- Sub-block 3: $dz3, dy3$ and $Tmat3$.

At the end of each sub-block the user will always have the chance to reintroduce the data if an error has been made. This concludes the part of the program asking for all the data required for Block 1.

4.3.2.3. Input of Block 2: determination of products of inertia

The input of this block is very similar to the previous one (see Listing 4.5), with the main difference being that there is an emphasis in making sure that the *CubeSat* geometrical centre is contained within the rotational axis of the pendulum. This is vital for completing the method described in section 3.2.3.. Evidently, yet another calibration will have to be made in order to account for the fact that we are now using a support to hold the *CubeSat* in a 45° angle. Following the calibration, the next steps are basically the same as in the previous block with the program instructing the user in the placement of the *CubeSat* making use of the function $AskPeriod$ (4.3.8.1.).

As usual at the end of this whole block, the option to repeat the inputs will be made should the user deem it necessary. All in all, we will store the values of the periods in three separate vectors containing in each cell the period of an individual experiment given in seconds):

- Experiment 2.1: $T1exp$.
- Experiment 2.2: $T2exp$.
- Experiment 2.3: $T3exp$.

Listing 4.5: The script responsible for Block 2

```
            fprintf('Measure of Izy (Experiment 2.1)');
            fprintf('Place the satellite on the support with the ...
                X axis parallel to the platform. \n Make sure ...
                that the selected coordinate origin is contained ...
                within the rotation axis. \n');
            T1exp=AskPeriod('Write the periods of the satellite ...
                on the support. ','Input a period, if done input ...
                0: \n');
            fprintf('Measure of Izx (Experiment 2.2)');
            fprintf('Place the satellite on the support with the ...
                Y axis parallel to the platform. \n Make sure ...
                that the selected coordinate origin is contained ...
                within the rotation axis. \n');
            T2exp=AskPeriod('Write the periods of the satellite ...
                on the support. ','Input a period, if done input ...
                0: \n');
            fprintf('Measure of Iyx (Experiment 2.3)');
            fprintf('Place the satellite on the support with the ...
                Z axis parallel to the platform. \n Make sure ...
                that the selected coordinate origin is contained ...
                within the rotation axis. \n');
            T3exp=AskPeriod('Write the periods of the satellite ...
                on the support. ','Input a period, if done input ...
                0: \n');
            repeat=input('In the event of needing to repeat this ...
                section please write 0: \n');
        end
```

4.3.3. Input of simulation variables

This section is entered if the user wants to simulate an experiment instead of performing a real measure. The overall aim of this section is to obtain the variables necessary to perform the method in the same way as if it were a real experiment. In order to do so, the program needs to know two separate pieces of information: the periods of the experiment, as seen before, and the parameters of the experiment itself.

4.3.3.1. Input of experiment variables

The purpose of this section is to offer the users as much freedom as possible when designing the parameters of the experiment they want to simulate. These variables affect the overall procedure of the experiment and can be tailored to suit specific real-life counterparts. Alternatively, the users can always choose not to input the variables themselves, and just use the default values stored in the program. These values have been chosen based on a work previously done (see reference (1)), and correspond to the pendulum already designed. The code responsible to do this can be found in Listing 4.6.

Listing 4.6: Code showing the default input for simulation variables

```
end
Initial=2;
prec=-5;%Order of the precision for the time measurement ...
    equipment
precm=-5;%Order of the precision of the mass measurement
siser=10^-7;%The sistematic error of the time ...
    measurement equipment
siserm=10^-7;%The sistematic error of the mass measurement
b=0.5;%Torsion Coefficient N*m^2
precb=-7;%Order of the precision for the torsion coefficient
siserb=10^-9;%Sistematic error for the torsion coefficient
Iempty=0.38825;%The moment of the empty pendulum
Isup=0.589;%The moment of the pendulum with the support
dy1=0.1;%The displaced distance in Block 1 section 1
dx1=0.1;%The displaced distance in Block 1 section 1
dz2=0.1;%The displaced distance in Block 1 section 2
dx2=0.1;%The displaced distance in Block 1 section 2
dy3=0.1;%The displaced distance in Block 1 section 3
```

Either way, the list of obtained variables is the following:

- b,$precb$ and $siserb$: The torsion constant of the pendulum, its random error and systematic error. The variables b and $siserb$ are given in N·m^2 and the random error $precb$ in logarithmic units.
- $prec$,$siser$: The precision of the period measurement and its systematic error. Note that $prec$ is then transformed to logarithmic units because, when measuring the periods, it will be introduced by using power units.
- $Iempty$: The moment of inertia of the empty pendulum, in kg m^2.
- $Isup$: The moment of inertia of the pendulum with the 45° support placed on it, in kg m^2.
- ni: The amount of times period measurement is performed.
- $precm$,$siserm$: The precision of the mass measurement and its systematic error. Note that $precm$ is then transformed to logarithmic units because, when measuring the masses, it will be introduced by using power units.

4.3.3.2. Input of Inertia Tensor data

In this section, the program will present three different ways to introduce the input the data of an experiment: reading the data from a file, manually entering the data, and using the default example. These data correspond to the inertia tensor characteristics of the *CubeSat* we simulate. In all three ways the end-result will be the same:

- *TensT*: The inertia tensor with respect to a chosen frame of reference, in kg m^2. In *CubeSats* its origin usually corresponds to their geometric centre.
- *CoordT*: The coordinates of the centre of mass with respect to the same coordinate origin as *TensT*, in meters.

- m: The mass in kg of the *CubeSat* in question.

For each way of introducing these variables, a specific function has been written:

- $ReadFile$ (function 4.3.8.3.): for reading data from a file;
- $ManualInput$ (function 4.3.8.4.): if the users wish to enter data manually;
- $AnalyticalInput$ (function 4.3.8.2.): if the users decide to use the default example of a cube with an off-center spherical hole in it.

4.3.4. Simulation: Calibration results

The results of the calibration are used in every step of the method. Therefore, before simulating the experiment, we need to have calculated the inertia tensor of the empty pendulum and the pendulum with the support placed on it. In the event the user has chosen to input real data, the calculation is pretty straight forward (see Listing 4.7). Having two vectors with the periods, one for the empty pendulum named $T0$ and the other for the pendulum with the 45° support named $T0sup$, we calculate their respective averages and perform the inverse of the equation 3.1 to obtain the moment of inertia.

Listing 4.7: Calculation of empty pendulum moments

```
1          br=b+(-1+2*rand)*10^precb+siserb;
2          T1=sqrt(Iempty/br)*2*pi;
3          T0=T1*ones(ni,1)+(-1+2*rand(ni,1))*10^prec+siser;% ...
               Period of the empty pendulum [s]
4          T1sup=sqrt(Isup/br)*2*pi;
5          T0sup=T1sup*ones(ni,1)+(-1+2*rand(ni,1))*10^prec+siser;
```

If the user chooses to simulate the experiment (see Listing 4.1) the program needs first to generate the period vectors $T0$ and $T0sup$. In order to do this, the program starts with the actual inertia moments $Iempty$ and $Isuport$, calculates their period using the inverse of Equation 3.1, and then introduces a random and a systematic error. This procedure will be explained further in section 4.3.5.. This error will be introduced ni times therefore generating our objective vectors. Finally, the function $CalComp$(4.3.8.6.) can be applied and the simulated moments of the empty pendulum extracted.

Listing 4.8: list:emptysim

```
1      T0m=mean(T0);
2      I0m=CalComp(T0m,b);
3      T0supm=mean(T0sup);
4      I0supm=CalComp(T0supm,b);
```

Once this step is done, the following calculations will be equal to the ones corresponding to the real case.

4.3.5. Simulation: Calculus of Periods and introduction of error

The main aim of this section of the program is to ~~actually~~ simulate the procedure of the experiment as accurately as possible. This will mainly consist of taking the tensor data of our simulated *CubeSat* and the experimental data, then calculate the periods for the different resulting configurations. In order to make our simulation as similar as possible to the conditions of an actual experiment, errors will be introduced in our measurements. These errors will be both random and systematic, and will affect the time measurement equipment, the value of the torsion constant and the mass of our *CubeSat*.

With the purpose of introducing the random error in all of these measurements we will select a random number in the range of the precision. This calculation can be done with the *rand* function of MATLAB, which gives us a random number in the interval [0,1] with a uniform distribution. The random error can also be negative so we will multiply the value of the function by two and subtract one in order to achieve a random number in the interval [-1,1]. Finally, we will then multiply this value by the maximum error the precision gives us. With the random error done we will add (or subtract, depending on the sign) the systematic error. In Listing 4.9, the original value is b, the order or the random error is given by $precb$ and the systematic error is $siserb$.

Listing 4.9: Introduction of error in the torsion constant b

```
1          br=b+(-1+2*rand)*10^precb+siserb;
```

4.3.5.1. Period simulation for a experiment block

The aim when simulating a block is to produce a matrix in the same format as when introducing variables from the real experiment. In this example we will use the simulation of Block 1.1 (I_{zz},3.2.2.1.) which is analogous to the one explained in 4.3.2.2..

The resulting matrix will be composed of 3 rows, each containing the periods of a subsection of the experiment, with length assigned by the variable ni (Introduced in 4.3.3.1.); which corresponds to the number of times each period is measured for a given configuration of the *CubeSat*. In order to calculate these periods we will first calculate the moment of inertia of the whole configuration.

The total moment of inertia will be the result of adding the introduced moment of the empty pendulum (I_0) plus the moment of the *CubeSat* I_{CM}, displaced with Steiner's theorem (in experiment 1.1.1 $dy1$ and $dz1$). We will also make use of the centre of mass coordinates $X1$ and $Y1$. This results in the following equation:

$$I_{\mathrm{Tot}} = I_0 + I_{\mathrm{CM,zz}} + [(dy1 - Y1)^2 + X1^2] \cdot m \tag{4.3}$$

This expression will be used in every experiment, changing the moment of the *CubeSat* and the displacement as the configuration of the *CubeSat* change. Using the expression 3.1 and isolating the period, we can obtain the exact period this configuration will yield. The code section in Listing 4.3.5.1. performs this calculation for the case 1.1.1:

Listing 4.10: list:sim

```
T1=sqrt((Iempty+TensTCM(3,3)+((dy1-CoordT(2))^2+...
    CoordT(1)^2)*mr)/br)*2*pi;
```

It deserves mentioning that in every case we need to perform a calculation analogous to that of the expression 4.3.5.1., we will use the variables br and mr, which include errors in their value (Calculus for br done in 4.3.5.). Once we have the period, we need to generate a vector of different values around this period. We will do this by introducing the random error (different for each vector component) given by our precision, and also adding a fixed (systematic) error to all of the vector($T1sim$). The random error will follow a uniform distribution which is given by the function $rand$ of MATLAB. All in all, for the experiment sub-block 1.1, we will obtain a matrix called $Tmat1$ with the three subsections corresponding to 3 rows.

Listing 4.11: Creation of the matrix $Tmat1$

```
T1=sqrt((Iempty+TensTCM(3,3)+((dy1-CoordT(2))^2+...
    CoordT(1)^2)*mr)/br)*2*pi;
T1sim=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
T2=sqrt((Iempty+TensTCM(3,3)+((dx1+CoordT(1))^2+...
    CoordT(2)^2)*mr)/br)*2*pi;
T2sim=T2*ones(1,ni)+rand(1,ni)*10^prec+siser;
T3=sqrt((Iempty+TensTCM(3,3))/br)*2*pi;
T3sim=T3*ones(1,ni)+rand(1,ni)*10^prec+siser;
Tmat1=[T1sim;T2sim;T3sim];
```

For the other sub-block of block 1 (calculation of moments of inertia) we will have a corresponding matrix. At the end of the process, we will have the following variables:

- Sub-block 1: $Tmat1$.
- Sub-block 2: $Tmat2$.
- Sub-block 3: $Tmat3$.

This method is also performed in a similar way for the experiments of Block 2 (calculation of products of inertia) with the calculation of the period and the introduction of error.

Listing 4.12: Simulation of experiment 2.1

```
        Iprot=RotMat(0,0,pi/4,TensT);
        T1=sqrt((Iprot(3,3)+Isup)/br)*2*pi;
        T1exp=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
```

For Block 2 the function $RotMat$ (4.3.8.5.) will be used in order to compute the inertia tensor of the *CubeSat* with respect to its geometric centre when we rotate it.

Repeating this process for the three sub-blocks of Block 2 will result in three separate vectors, containing the simulated periods for the three sections.

- Sub-block 1: $T1exp$.

- Sub-block 2: $T2exp$.
- Sub-block 3: $T3exp$.

At this point both pathways of the program converge to the same point and we can start using the tensor calculation procedure explained in 3 and review the results.

4.3.6. Inertia Tensor Calculation Method

This section of the code is devoted to calculating the moments from the calculated/input periods applying the method explained in 3. We must also make use of the experiment parameters introduced, as the code demands it.

One of the main focuses of this section of the program is to mitigate any errors which may have been made in the period measurements. This is the reason why various periods (ni) are provided for each subsection of the method in the form of vectors. Having a large number of measurements can help us mitigate the random error that inescapably occurs in the use of equipment. As a result, the first operation when given a period vector is to calculate its mean. Doing this step as early as possible will prevent the error amplification in the method.

Bear in mind that the user could have made a different number of measurement on different subsections in the event of a real experiment. This results in a variable ni for each section as opposed to a constant ni if the program is simulating a experiment. Even though this will not affect the method, for the sake of clarity and simplicity we will refer to ni as a constant for the following explanations.

As a starting point of this section the variables we will have are:

- $Tmat1, Tmat2, Tmat3$: The three matrices of size $3 \times ni$ containing the periods [S] of Block 1, which correspond to sub-block 1, sub-block 2 and sub-block 3 respectively.
- $dy1$,$dx1$,$dx2$,$dz2$,$dx3$,$dz3$: The displacement the *CubeSat* has suffered in each subsection of Block1 [m]. The last number of the variable name indicates the corresponding section.
- $Texp1, Texp2, Texp3$: The three vectors of length ni containing the periods[S] of Block 2, which correspond to sub-block 1, sub-block 2 and sub-block 3 respectively.
- b: The torsion constant of the pendulum in N·m^2 .
- m: The mass in kg of the *CubeSat* in question.
- $I0m$,$I0supm$: The moments of the empty pendulum and the pendulum with the support[kg m^2], obtained from the calibration steps.

Whenever a moment of inertia has to be derived from a period, the function $CalComp$ will be used(4.3.8.6.). For the simulation case the variables b and m are used, instead of br and mr, in order to stay as faithful as possible to an actual experiment in which the real value of these variables is not known.

The two code fragments shown and explained will be the ones corresponding to Block 1 sub-block 1 and Block 2 sub-block 1, whose equations are explained in 3.2.2.1. and 3.16 respectively.

4.3.6.1. Calculus of Block 1 sub-block 1

In this portion of the code the calculation corresponding to the experiment 1.1(3.2.2.1.) will be reviewed. As a reminder, here is the system of equations to solve and isolate variables:

$$\begin{cases} I_{1zz} = I_{CM,zz} + m[(dy - Y_0)^2 + X_0^2] \Longrightarrow I_{1zz} = I_{3zz} + m(dy^2 - 2Y_0 dy) \\ I_{2zz} = I_{CM,zz} + m[(dx + X_0)^2 + Y_0^2] \Longrightarrow I_{2zz} = I_{3zz} + m(dx^2 + 2X_0 dx) \\ I_{3zz} = I_{CM,zz} + m(X_0^2 + Y_0^2) = I_{CG,zz} \end{cases} \tag{4.4}$$

The variables used will be $Tmat1$, $I0m$, b and m. The first thing to do is to calculate the average of the period of each row of $Tmat1$, we will then use the function $CalComp$ to derive their corresponding moments of inertia and, finally, subtract the value of the empty pendulum $I0m$. This procedure will give us the moment of inertia of the *CubeSat* on that subsection. The following table shows the correlation between the moments of inertia in Equation **??**, and the variable names used in the program. Additionally, it also shows the calculation performed in the script in order to calculate them.

Name in Expression **??**	Name in script	Calculation in script
I_{1zz}	$I1$	$CalComp(mean(Tmat1(1,:)),b) - I0m$
I_{2zz}	$I2$	$CalComp(mean(Tmat1(2,:)),b) - I0m$
I_{3zz}	$I3$	$CalComp(mean(Tmat1(3,:)),b) - I0m$
dy	dy	$dy = dy1$
dx	dx	$dx = dx1$

Table 4.1: Variable nomenclature for experiment 1.1. The syntax $Tmat1(X,:)$ means that the program selects the row X from the matrix $Tmat1$.

After obtaining the values of I_{1zz},I_{2zz} and I_{3zz} it is a matter of isolating the unknown variables and computing their values as seen in the expressions **??**, 3.7, 3.6 and 3.8. The moments of inertia will be stored in the matrices *CMtens* and *CGtens* which correspond to the variables I_{CM} and I_{CG} on sub-block 3.2.2.1.. The following table (4.2) shows the name of the unknown variables in both the expressions of 3.2.2.1. and the script, as well as the equations used to reach them.

This calculation is repeated for the remaining two sub-block of Block 1 but changing the procedure accordingly. All in all, at the end of the calculations in Block 1, we will have the moments of inertia and the coordinates of the centre of mass. The name of the variables and their contents are:

- *CGtens*=Diagonal matrix with the principal moments of inertia with respect to a frame of reference centred in the geometric centre/coordinate origin, in kg m^2.
- *CMtens*=Diagonal matrix with the principal moments of inertia with respect to a frame of reference centred in the centre of mass, in kg m^2.

- $X01, Y01, X02, Z02, Y03, Z03$=Calculated coordinates of the centre of mass, in m. The last number corresponds to the sub-block they have been computed in. There is some overlap, which will be reviewed further along.

Step $n°$	Unknown in 3.2.2.1.	Isolation express -ion in 3.2.2.1.	Unknown in script	Isolation expression in script
Step 1	X_0	3.7	$X01$	$X01 = (I2 - I3)/(m*dx*2) - dx/2$
Step 2	Y_0	3.6	$Y01$	$Y01 = ((I1 - I3)/(m*dy*2) - dy/2)*(-1);$
Step 3	$I_{\mathrm{CM,zz}}$	3.8	Izz	$Izz = I3 - m*(X01^2 + Y01^2) = CGtens(3,3)$
Step 3	$I_{\mathrm{CG,zz}}$	$I_{\mathrm{CG,zz}} = I_{3zz}$	$CGtens(3,3)$	$CGtens(3,3) = (I3)$

Table 4.2: Calculation step description and variable nomenclature.

We are now missing only the products of inertia in both tensor matrices, which means it is time to perform Block 2 calculations.

4.3.6.2. Calculus Block 2 sub-block 1

In this section the task at hand is to perform the necessary calculations in order to derive the products of inertia. As for the design of the calculations (3.16), we will only fill the tensor components with respect to the coordinate origin/geometric centre.

Similarly to the calculations in Block 1, we start with a vector of periods for our configuration ($T1exp$ for this particular example). Firstly, the mean is extracted from this set of periods in order to keep amplification of errors at bay. Once the average is extracted, we can compute the corresponding moment of the total configuration with $CalComp$ and, eventually, subtract the value of the pendulum moment ($Isupm$). Once we reach this step, the expression 3.24 can finally be used to find the moment of the *CubeSat*. The following table details the variables used and their corresponding name in section 3.2.3. of the theoretical fundamentals.

With this three variables we can use the expression 3.24 in order to fill the corresponding positions of *CGtens*. With $CGtens(2,3)$ and $CGtens(3,2)$, corresponding to the variable $I_{\mathrm{CG,zy}}$ of expression 3.24.

$$I_{\mathrm{Rot1}} = R\, I_{\mathrm{CG}}\, R^T \longrightarrow I_{\mathrm{Rot1,zz}} = \frac{1}{2}(I_{\mathrm{CG,xx}} - 2I_{\mathrm{CG,yz}} + I_{\mathrm{CG,zz}}) \tag{4.5}$$

Name in Expression 3.24	Name in script	Calculation in script
$I_{\mathrm{Rot1,zz}}$	$I1$	$CalComp(mean(T1exp), b) - I0supm$
$I_{\mathrm{CG,xx}}$	$CGtens(1,1)$	Calculated in Block 1 (3.2.2.1.)
$I_{\mathrm{CG,zz}}$	$CGtens(3,3)$	Calculated in Block 1 (3.2.2.1.)

Table 4.3: Variable name correlation table for experiment 2.1.

Listing 4.13: Code correspondent to expression 3.24

```
Iprot=RotMat(0,0,pi/4,TensT);
T1=sqrt((Iprot(3,3)+Isup)/br)*2*pi;
T1exp=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
```

Repeating this step for the two remaining sub-blocks of Block 2, the matrix *CGtens* will be completely determined. At this point of the program, the products of inertia of *CMtens* are still missing. We deal with this issue in the last parts of our code.

4.3.6.3. Auxiliary calculations for centre of mass and I_{CM}

The final step of the method consists of arithmetic operations to complete the calculus of the remaining unknown variables. Firstly, the coordinates of the centre of mass have to be known, bearing in mind that, at this point, we have two values for each of the three coordinates. We will solve this by calculating the mean between the two values and using the result as the centre of mass position. The list of variables is as follows:

- $X01, Y01$=The coordinates of the CM in the X and Y axis [m] calculated in Block 1 Sub-block 1.
- $X02, Z02$=The coordinates of the CM in the X and Z axis [m] calculated in Block 1 Sub-block 2.
- $Z03, Y03$=The coordinates of the CM in the Z and Y axis [m] calculated in Block 1 Sub-block 3.

The calculations have been made as follows with the final value of the coordinates stored in the vector of size 3 *CoordV* (see Listing 4.14).

Listing 4.14: Calculation of the coordinates of the CM

```
Xcm=mean([X01,X02]);
Ycm=mean([Y01,Y03]);
Zcm=mean([Z02,Z03]);
```

We now have a complete *CGtens* and accurate coordinates of the CM(*CoordV*). At this point, we would be ready to apply the expression 3.32 in order to complete *CMtens*. However, before we start calculating anything, let us recall that the ~~principal~~ moments of inertia of *CMtens* have already been computed. Calculating them again would be redundant and, even worse, could result in an amplification of errors. Therefore, expression 3.32 will be applied for only the secondary vectors. The resulting equation and variable correlation will be as follows:

Variable name in expression 3.32	Variable name in the script
I_{CG}	CGtens
I_{CM}	CMtens
X_0	$CoordV(1)$
Y_0	$CoordV(2)$
Z_0	$CoordV(3)$

Table 4.4: Table of variable name correlation with expression 3.32

The expression used in the program will be a slightly modified version of Equation 3.32. The variable names used in the following expression will be the ones used in the calculations of 3, with the exception of I_{CM1}, which corresponds to the diagonal matrix *CMtens* containing the principal moments with respect to the CM.

$$I_{\mathrm{CM}} = I_{\mathrm{CG}} - Steiner + I_{\mathrm{CM1}} = \begin{bmatrix} 0 & I_{\mathrm{CG,xy}} & I_{\mathrm{CG,xz}} \\ I_{\mathrm{CG,yx}} & 0 & I_{\mathrm{CG,yz}} \\ I_{\mathrm{CG,zx}} & I_{\mathrm{CG,zy}} & 0 \end{bmatrix} - m \cdot \begin{bmatrix} 0 & -X_0 Y_0 & -X_0 Z_0 \\ -Y_0 X_0 & 0 & -Y_0 Z_0 \\ -Z_0 X_0 & -Z_0 Y_0 & 0 \end{bmatrix} + I_{\mathrm{CM1}} \tag{4.6}$$

The codes section that performs this calculation is:

Listing 4.15: Calculation of secondary moments of *CMtens* with expression 4.6

```
1     CMtens=CGtens-diag(diag(CGtens))-m*[0, -Xcm*Ycm, -Xcm*Zcm;
2                        -Xcm*Ycm, 0, -Ycm*Zcm;
3                        -Xcm*Zcm, -Ycm*Zcm, 0]+diag(diag(CMtens));
```

At this point the method for inertia tensor calculation has been completed and the results are shown. The results are:

- *TensCM*: The 3×3 inertia tensor of the object with respect to its centre of mass in kg m^2.
- *TensCG*: The 3×3 inertia tensor of the object with respect to the chosen coordinate origin in kg m^2.
- *CoordV*: The X, Y, Z coordinates of the centre of mass in m.

4.3.7. Presentation of results and Auxiliary functions

Once the program has finished the calculations described, the results are presented to the user. This is divided into two sections; the first one is common to both the real experiment and the simulation pathways and the second section is specific to the simulation path. Once the results are presented, the user will have the option to save them using the function $WriteFile$ (4.3.8.7.) and also to repeat the whole program.

4.3.7.1. Presentation of results: Common pathway

The focus of this first section is to present the user with the experiment results in as clear a way as possible. The code is pretty simple, as we use the MATLAB function *fprintf* to display the variables we have been calculating in the section 4.3.6.. The displayed variables will be:

- *TensCM*: The 3×3 inertia tensor of the object with respect to a frame of reference centered in its centre of mass, in kg m^2.
- *TensCG*: The 3×3 inertia tensor of the object with respect to a frame of reference centered in the chosen coordinate origin, in kg m^2.

- *CoordV*: The X, Y, Z coordinates of the centre of mass, in m.

Additionally a diagonalization of both the *TensCM* and the *TensCG* will be performed with the function eig incorporated in MATLAB. This will give us the principal axes of inertia with respect to both the coordinate origin and centre of mass, and their respective values. The new variables will be

- VCG,VCM=The 3×3 matrices whose columns correspond to the three right hand eigenvectors, which represent the principal axes of inertia in the coordinate origin and in the centre of mass respectively.
- DCG,DCM=The 3×3 diagonalized matrices that correspond to the inertia tensors with respect to the principal axes centred in the coordinate origin and in the centre of mass [kg m^2].

Once the results are shown, the users will be presented with the option to see the tensor with respect to a frame of reference of their choosing (see Listing 4.16) .If the user chooses to view the tensor with respect to a frame of reference centred in a new point, the program will ask to introduce its coordinates [X, Y, Z] and store them in a variable called $Newpoint$. We will then calculate the displacement vector between this point and the centre of mass, and store it in a new variable called $NewVector$. Finally, with this new displacement vector, Steiner's theorem can be applied (2.1.2.) and the new tensor (sensibly named $NewTens$) is computed.

Listing 4.16: Calculation of a inertia tensor with respect to a new point using Steiner

```
1          NewVector=CoordV-Newpoint;
2          NewTens=CMtens+m*[NewVector(2)^2+NewVector(3)^2, ...
               -NewVector(1)*NewVector(2), -NewVector(1)*NewVector(3);
3                          -NewVector(1)*NewVector(2), ...
                               NewVector(1)^2+NewVector(3)^2, ...
                               -NewVector(2)*NewVector(3);
4                          -NewVector(1)*NewVector(3), ...
                               -NewVector(2)*NewVector(3), ...
                               NewVector(2)^2+NewVector(1)^2];
```

If the user wants to see this tensor *NewTens* with respect to a set of rotated axis the program will ask for the three Euler angles (2.1.3. and store them in a vector called *NewAngles* ([α, β, γ] in degrees). Then, the function *RotMat* (4.3.8.5.) is called to perform the calculations and the new tensor is displayed. Additionally, a diagonalization is made to see the principal axes of inertia and their moment with respect to this new reference system.

In addition to this, the users can also save the calculated result using the function *WriteFile* (4.3.8.7.) in a location of their choosing in the computer.

4.3.7.2. Results Specific to Simulation

In the event the user has simulated an experiment, additional computations are made. The goal of these computations is to compare the original tensor data with the data extracted from our method. As a result, the users can see if the experiment has performed to their standards.

The values used will be *TensCM*, *TensCG* and *CoordV*, which will be compared to their original counterparts: *TensT*, *TensTCM* and *CoordT*. The first comparison made is a calculation of the relative error; for this, each component of the simulation variables will be matched with those of the original ones, yielding a matrix/vector with the relative error stored in each component. Furthermore, the program will also compute the absolute error and store it in matrix form. The matrices of both these errors will be displayed to the user for reviewing.

4.3.8. Functions used

In this section we will review the functions referenced in the previous sections, we explain how they work, as well as their different inputs and outputs.

4.3.8.1. AskPeriod

This function (see Listing **??**) is used every time a period is needed. As seen before in our process, the effects of random errors are smoothed out by making several consecutive measurements with the same configuration. The function will write an opening statement describing the nature of the required periods, and then ask for period data until the user inputs 0:

Listing 4.17: The *AskPeriod* function script

```
1      while (finished≠0)
2          realpos=0;
3          dev=0;
4          while((realpos==0)&&(dev==0))
5              PeriodV(i)=input(FollowingStatement);%Store all the ...
                   periods in a vector
6              real=isreal(PeriodV(i));
7              pos=(PeriodV(i)≥0);
8              realpos=real*pos;
```

This function also checks if the input is real and positive, and discards it otherwise. This is done with the variables *real* and *pos* and a simple multiplication, so that the answer is accepted only if both requirements are met.

Furthermore, if the user has introduced more than 3 periods the program checks that, for every new period, the deviation does not surpass the standard deviation of the whole vector. In the event that it is greater, the user has the chance to introduce the value again or delete it. Inputs:

- *InitalStatemet*: String type variable that is typed once. Usually describing the nature of the expected data.
- *FollowingStatemet*: String type variable that is typed every time the program needs an input.

Outputs:

- *PeriodV*: Vector containing all the valid input periods.

4.3.8.2. *AnalyticalInput*

The purpose of this function is to provide analytical tensor data from a calculated example. In order to check every function of the program, we needed an example with an offset centre of mass. That is why we have chosen a homogeneous cube with a spherical hole in it. We have chosen the coordinate origin to be in the geometrical centre of the cube. For the calculation of the inertia tensor firstly we need to calculate the tensor of a cube and a sphere(Which radius is a quarter of the cubes length) separately.

Luckily, the calculation of the inertia tensor of a cube, with respect to a frame of reference with origin in its centre and axes parallel to its edges, is very simple because the chosen axes are principal axes of inertia. In addition, because of the symmetric geometry of the cube, all three principal moments of inertia are equal. The following equation presents the calculation for a cube of size a and density ρ.

$$I_{\text{cube,xx}} = \rho \int_{-a/2}^{a/2}\int_{-a/2}^{a/2}\int_{-a/2}^{a/2}(y^2+z^2)dxdydz = \frac{a^5\,\rho}{6} \quad \Rightarrow I_{\text{cube}} = \frac{a^5\rho}{6}\mathbb{I} \tag{4.7}$$

Where $\mathbb{I}$ is the identity 3×3 matrix. Following that, we can calculate the inertia tensor for a sphere of radius $r = \frac{a}{4}$ and density ρ, with respect to a frame of reference with origin in its centre, and then displace it using Steiner's method as explained before(2.1.2.). In this case, it is more convenient to calculate the tensor using spherical coordinates.

$$I_{\text{sphere,xx}} = \rho \int_M (y^2+z^2)dm = \frac{8}{15}\pi\rho\left(\frac{a}{4}\right)^5 \quad \Rightarrow I_{\text{sphere}} = \frac{8}{15}\pi\rho\left(\frac{a}{4}\right)^5\mathbb{I} \tag{4.8}$$

We will then displace the sphere (I_{sd}) to the point $(-\frac{a}{4}, -\frac{a}{4}, -\frac{a}{4})$ and subtract it to the tensor of the cube (I_{c}), in order to obtain the variable that we have called *TensT*. This is one of the variables that will be returned:

$$TensT = I_{\text{c}} - I_{\text{sd}} \tag{4.9}$$

Which if we choose $a = 0.1$ m and $\rho = 2700$ kg/m^3 will yield the following matrix:

$$TensT = \begin{bmatrix} 0.0043 & 5.8533\cdot10^{-5} & 5.8533\cdot10^{-5} \\ 5.8533\cdot10^{-5} & 0.0043 & 5.8533\cdot10^{-5} \\ 5.8533\cdot10^{-5} & 5.8533\cdot10^{-5} & 0.0043 \end{bmatrix} \tag{4.10}$$

The coordinates of the centre of mass will be calculated by taking both objects as point masses but, in the case of the sphere, introducing a negative mass and performing a simple calculation.

Hence, the function will not need any input and will return the following outputs:

- *TensT*: The inertia tensor with respect to a frame of reference centred in a chosen origin, in kg m^2. In an 1U *CubeSat* this usually corresponds to its geometric centre.
- *CoordT*: The coordinates of the centre of mass with respect to the same frame of reference as *TensT*, in meters.
- m: The mass in kg of the *CubeSat* in question.

4.3.8.3. *ReadFile*

This function is used when we want to read the tensor data from a *.txt* file of our choosing. This file can be written by hand or also be stored from a previous experiment with the function *WriteFile* (4.3.8.7.). The files contain all the information needed to complete our calculation: the tensor with respect to the coordinate origin (*CGtens*), the tensor with respect to the mass centre (*CMtens*), the coordinates of the centre of mass (*CoordV*) and the mass of the *CubeSat*(m). These data will be written in the form of a 8×3 matrix in a text file, and the option will appear to store it in any place in the computer. The complete matrix with all the data is called *SaveMatrix*, and its structure is the following:

$$SaveMatrix = \begin{bmatrix} (... & CGtens & ...) \\ (... & CMtens & ...) \\ (... & CoordV & ...) \\ (0 & 0 & m) \end{bmatrix} \tag{4.11}$$

For our purposes we will only read *CGtens*, *CoordV* and m because that is the data we will export:

- *TensT*= *CGtens*: The inertia tensor with respect to a PGPframe of reference centred in a chosen origin, in kg m^2. In an 1U *CubeSat* this usually corresponds to its geometric centre.
- *CoordT*=*CoordV*: The coordinates of the centre of mass with respect to the same frame as *TensT*, in meters.
- m: The mass, in kg, of the *CubeSat* in question.

4.3.8.4. *ManualInput*

This function is pretty simple, as it consists on a series of questions prompted to the user asking for the tensor data. The function will check whether the introduced inertia tensor is positive, real and symmetric and repeat the question in the event it is not. As always the given outputs will be:

- *TensT*: The inertia tensor with respect to a PGPframe of reference centred in a chosen origin, in kg m^2. In an 1U *CubeSat* this usually corresponds to its geometric centre.
- *CoordT*: The coordinates of the centre of mass with respect to the same coordinate origin as *TensT*, in meters.
- m: The mass, in kg, of the *CubeSat* in question.

4.3.8.5. *RotMat*

This function makes use of the rotation matrix with Euler's angles in order to calculate a new inertia tensor with respect to a rotated set of axis (2.1.3.).

Listing 4.18: Calculus of the new rotated matrix

```
EulM=[cos(alpha)*cos(beta), ...
    cos(alpha)*sin(beta)*sin(gamma)-sin(alpha)*cos(gamma), ...
    cos(alpha)*sin(beta)*cos(gamma)+sin(alpha)*sin(gamma)
     sin(alpha)*cos(beta), ...
        cos(alpha)*sin(beta)*sin(gamma)+cos(alpha)*cos(gamma), ...
        sin(alpha)*sin(beta)*cos(gamma)-cos(alpha)*sin(gamma)
     -sin(beta), cos(beta)*sin(gamma), cos(beta)*cos(gamma)
     ];
RotM=EulM*M*EulM';
```

The inputs are:

- $alpha$,$beta$,$gamma$=Euler's angles for the rotation, in radians.
- M=Original inertia tensor.

The function has a single output, which is the rotated inertia tensor *RotM*.

4.3.8.6. *CalComp*

This function applies the formula 3.1 to a whole set of periods, and returns a set of the same dimension with the corresponding moments.

Inputs:

- $T0$:A set of periods of dimension N in seconds.
- b: The torsion constant of the pendulum in N·m^2 .

The output will be a set of moments called Ip in kg m^2 of the same dimension N as the input.

4.3.8.7. *WriteFile*

This function works in a similar manner to *ReadFile* (4.3.8.3.), but in reverse: it writes the results of the experiment on a *.txt* file and saves it in a location of the users' choosing. The input variables are the following:

- *CGtens*: The inertia tensor with respect to frame of reference centred in a chosen coordinate origin, in kg m^2. In an 1U *CubeSat* this usually corresponds to its geometric centre.
- *CMtens*: The inertia tensor with respect to a frame of reference centred in the calculated centre of mass, in kg m^2.
- *CoordV*: The coordinates of the centre of mass with respect to the same coordinate origin as *TensT*, in meters.
- m: The mass in kg of the *CubeSat* in question.

All of these variables will be stored in a matrix that we have called *SaveMatrix* which, in turn, will be written on a *.txt* file. The matrix structure will be:

$$SaveMatrix = \begin{bmatrix} (... & CGtens & ...) \\ (... & CMtens & ...) \\ (... & CoordV & ...) \\ (0 & 0 & m) \end{bmatrix} \tag{4.12}$$

CHAPTER 5. CONCLUSIONS AND RESULTS

In this final chapter we summarize our main results and draw the main conclusions from our work.

5.1. Conclusions

Our work is based on a previous project in which an experimental setup aimed to the determination of the inertia tensor of *CubeSats* was designed (1). The actual design of the setup, which essentially consists of a torsion pendulum, was not modified. However, using theory of rotation dynamics, we introduced necessary improvements in terms of the expected performance and the actual operation of the device, which can be summarized as follows:

i) We proposed a method to determine the position of the centre of mass of the *CubeSat*, using the measured periods of the torsion pendulum. The position of the centre of mass, critical for its dynamics, was not determined in (1).

ii) We have confirmed the series of period measurements to be taken for specific positions of the *CubeSat*, which allow the determination of the moments of inertia which, after proper manipulation using matrix algebra, lead to the construction of the entire inertia tensor with respect to different frames of reference.

iii) We have developed a *CubeSat* code which serves multiple purposes: First, it guides the user of the experimental setup in following the necessary steps to complete the experiment. Second, using period measurements as inputs, it calculates and displays the results for the centre of mass and the inertia tensor components of the *CubeSat*. The detailed description of the code presented in this report may serve as an additional guide to the user of the experiment. Besides, it can facilitate a translation of the code (actually written in MATLAB) to another language which might eventually be more useful.

iv) In addition to being capable of processing real data (actual period measurements), the program is capable of simulating an experiment. In this type of simulations the program reads the tensor data we introduce, calculates the periods with the inverse of the method considered above, and introduces an error. Afterwards, the program follows the method as if it was a real experiment and shows the corresponding output. In service of realism, the program allows for the introduction of measurement errors in the simulation and shows the deviation from the original values this creates. This capability of the code is critical, as it will be used to test different configurations of the *CubeSat* in order to achieve the desired accuracy in the results. Eventually, this capability may also be used to refine the specific characteristics of the design of the torsion pendulum itself, or to help in the choice of the sensor req.

v) In order to prove that the method is sound and that the program works as expected, we have performed a simulation of the experiment for which, in ideal conditions, the tensor of inertia is known. In addition, we have asked the program to perform

the experiment of giving the tensor of inertia for a cube with the dimensions of a *CubeSat*, and a displaced spherical hole. Using realistic values for precision errors in measurement equipment, the maximum relative error in the principal moments has been of 0,7% and a maximum absolute error in the products of inertia of 15×10^{-5} kg m^2; whereas the maximum relative error in the calculation of the coordinates for the centre of mass has been of 2,5%.

All in all, the goals of the project have been achieved. This program provides a viable platform to perform the desired experiment and can also help in testing and simulation of future experiments. Once this method is implemented it can provide a relatively accessible pathway to accurate empirical inertia tensors of *CubeSats*, which will in turn improve the command and control capabilities of these devices.
Given the increasingly number of launched *CubeSats*, and the necessity of improving their attitude control for many interesting applications, the results of the present work and the actual construction and operation of the experimental setup will prove very useful to help *CubeSats* extend their capabilities, and be used in projects currently restricted to the context of high-budget missions.

5.2. Future improvements of the program

Additional improvements in the code could be implemented in the future, aiming to make profit of the simulation capabilities of the program in a wider scope. For instance, prior to the simulation,the program could ask the user both about the experiment variables and the precision requirements of the final result. These would include tensor calculation target precision and precision for the location of centre of mass. The program would then calculate the deviation due to the systematic and random errors, and compare it to the target precision. Depending on the result two cases would arise.

i) Case for systematic deviation greater than target precision: In this case the deviation caused by the systematic errors in the different measurement equipment would cause the result to never be within the requirements. As a result, the program would advise for better measurements.

II) Case for systematic deviation lower than target precision: For this case the result could be within the bounds. However, the random error of the equipment is still to be taken into account. The advantages of using the mean of several (ni) measures as the final value in each subsection, results in the final deviation decreasing inversely to the number of measurements made (ni). The program would therefore try the experiment with different ni values and propose one high enough that ensures that the final result is within our bounds but not so high that the experiment becomes unpractical.

Another important development should be to perform a rigorous error analysis. This is very relevant in order to determine the sources of systematic and random errors. It must be kept in mind that the errors will be different for Block 1 (*Principal moments*) and Block 2 (*Inertia products*) due to the presence of the support in Block 2. Then, it is very likely that

the accuracy of Block 1 will be better. The best way to reduce this difference would be to build a support as light as possible.

In terms of hardware, of course, the actual construction of the torsion pendulum and its calibration and exploitation are expected to be done in the near future. A sensor system aimed to detect the actual configuration of the *CubeSat* and warn the user if such configuration does not match the one expected by the software will also be designed and constructed.

BIBLIOGRAPHY

[1] Marc Gabarro Alsina. *A system to determine the inertia tensor of small satellites*.

[2] Jordi Puig-Suari, Scott Williams, and Roland Coelho. Cubesat design specification rev. 13. *The CubeSat Program, Cal Poly SLO*, 2014.

[3] David Morin. *Classical Mechanics*. Cambridge University Press, 2007.

[4] Jerry B. Marion. *Dynamics of Systems and Particles*. Academic Press, 1965.

APPENDICES

APPENDIX A. DETAILED PENDULUM DIAGRAMS

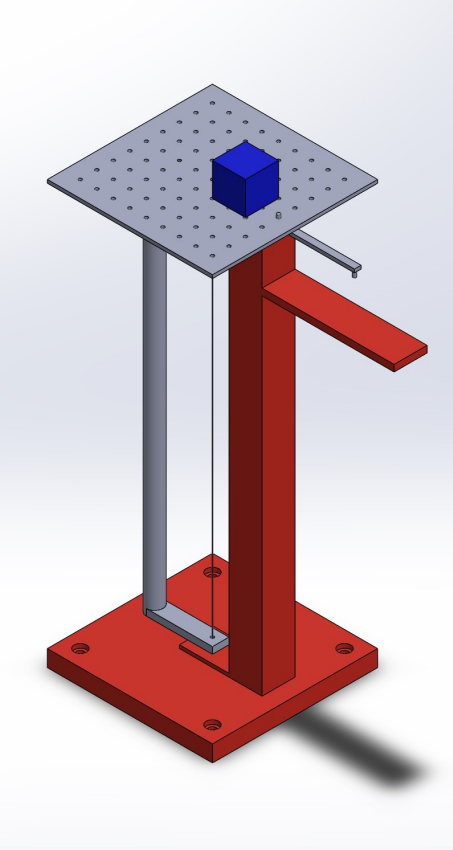


Figure A.1: Pendulum in neutral position.

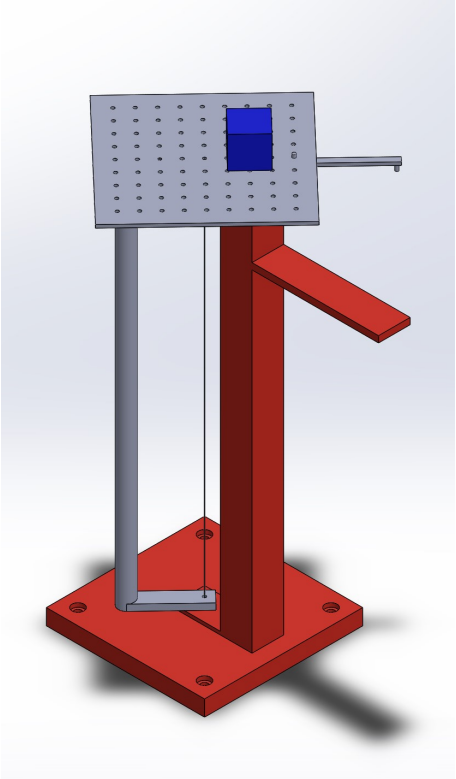


Figure A.2: Example of pendulum positioned to one side.

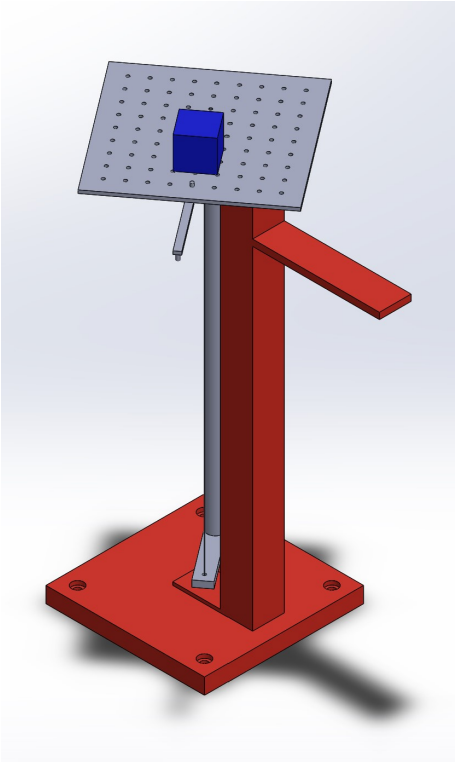


Figure A.3: Example of pendulum oscillating to the other side.

APPENDIX B. VARIABLE TABLE

The following table lists the variable used in the program and a brief description.

Name in script	Function
absErrorCM	1X3 vector storing the absolute error in center of mass coordinates
absErrorMaTCG	3X3 matrix storing the absolute error of I_{CG}
absErrorMaTCM	3X3 matrix storing the absolute error of I_{CM}
b	Torsion constant
br	Torsion constant with error for simulations
CGtens	3X3 matrix with calculated tensor with respect to the CG I_{CG}
CMtens	3X3 matrix with calculated tensor with respect to the CM I_{CM}
CoordT	1X3 vector with original coordinates of the center of mass
CoordV	1X3 vector with calculated coordinates of the center of mass $[X_0, Y_0, Z_0]$
DCG	Diagonalized I_{CG} 3X3 matrix
DCM	Diagonalized I_{CM}3X3 matrix
Dnew	Diagonalized 3X3 matrix with respect to a new coordinate system
dx	Distance the *CubeSat* is moved in any given experiment along the X axis
dx1	Distance the *CubeSat* is moved in experiment 3.2.2.1. along the X axis
dx2	Distance the *CubeSat* is moved in experiment 3.2.2.2. along the X axis
dy	Distance the *CubeSat* is moved in any given experiment along the Y axis
dy2	Distance the *CubeSat* is moved in experiment 3.2.2.2. along the Y axis
dy3	Distance the *CubeSat* is moved in experiment 3.2.2.3. along the Y axis
dz	Distance the *CubeSat* is moved in any given experiment along the Z axis
dz2	Distance the *CubeSat* is moved in experiment 3.2.2.2. along the Z axis
dz3	Distance the *CubeSat* is moved in experiment 3.2.2.3. along the Z axis
ErrorCM	1X3 vector storing the relative error in center of mass coordinates
ErrorMaTCG	3X3 matrix storing the relative error of I_{CG}
ErrorMaTCM	3X3 matrix storing the relative error of I_{CM}
I0m	Calculated moment of the empty pendulum
I0supm	Calculated moment of the empty pendulum with the support
I1	Calculated moment of a subsection of Block 1
I2	Calculated moment of a subsection of Block 2
I3	Calculated moment of a subsection of Block 3
Iempty	Original moment of the empty pendulum
Initial	Variable of control for input of parameters of experiment
Iprot	3X3 matrix storing a rotated inertia tensor
Isup	Original moment of the empty pendulum with the suppport
Ixx	Calculated $I_{CM_{xx}}$ in 3.2.2.3.
Iyy	Calculated $I_{CM_{yy}}$ in 3.2.2.2.
Izz	Calculated $I_{CM_{zz}}$ in 3.2.2.1.
m	Mass of the *CubeSat*
mr	Mass of the *CubeSat* with error of simulations
Newpoint	New point of reference
NewTens	New inertia tensor with respect to a new point in 3X3 matrix form
NewVector	1X3 vector from the calculated center of mass to the new point

ni	Number of consecutive periods made
options	Option of original tensor data input for simulations
prec	Random error of time measurement equipment
precb	Random error in torsion constant
precm	Random error in mass
repeat	Option to repeat program
sim	Option to simulate or perform a real experiment
siser	Systematic error in time measurement equipment
siserb	Systematic error in torsion constant
siserm	Systematic error in mass
T0	Vector of simulated periods of the empty pendulum
T0m	Mean of the simulated period of the empty pendulum
T0sup	Vector of simulated periods of the empty pendulum with the support
T0supm	mean of the simulated period of the empty pendulum with the support
T1	Calculated period of subsection 1 of any section of block 1
T1exp	1Xni vector with calculated periods of section 2 of block 2 with error
T1sim	1Xni vector with calculated periods of any subsection of block 1 with error
T1sup	Calculated period of the empty pendulum with the support
T2	Calculated period of subsection 2 of any section of block 1
T2exp	1Xni vector with calculated periods of section 2 of block 2 with error
T2sim	1Xni vector with calculated periods of any subsection 2 of block 1 with error
T3	Calculated period of subsection 3 of any section of block 1
T3exp	1Xni vector with calculated periods of section 3 of block 2 with error
T3sim	1Xni vector with calculated periods of any subsection 3 of block 1 with error
tensordisp	Option to view the tensor with respect to a new point
tensordisp1	Option to view the tensor with respect to a rotated system
TensT	3X3 matrix with original tensor with respect to origin for simulation
TensTCM	3X3 matrix with original tensor with respect to CM for simulation
Tmat1	3Xni matrix with periods for section 1 of block 1 3.2.2.1.
Tmat2	3Xni matrix with periods for section 2 of block 1 3.2.2.2.
Tmat3	3Xni matrix with periods for section 3 of block 1 3.2.2.3.
VCG	1X3 vector containing the eigenvalues of I_{CG}
VCM	1X3 vector containing the eigenvalues of I_{CM}
Vnew	1X3 vector containing the eigenvalues of the tensor in a new coord. system.
write	Option for writing the results in a file
X01	X coordinate of mass calculated in section 1 of block 1 3.2.2.1.
X02	X coordinate of mass calculated in section 2 of block 1 3.2.2.2.
Xcm	X coordinate of mass calculated from the mean
Y01	Y coordinate of mass calculated in section 1 of block 1 3.2.2.1.
Y03	Y coordinate of mass calculated in section 3 of block 1 3.2.2.3.
Ycm	Y coordinate of mass calculated from the mean
Z02	Z coordinate of mass calculated in section 2 of block 1 3.2.2.2.
Z03	Z coordinate of mass calculated in section 3 of block 1 3.2.2.3.
Zcm	Z coordinate of mass calculated from the mean

Table B.1: Table of variables

APPENDIX C. PROGRAM CODE

Listing C.1: Function for analytical example.

```
function [TensT,CoordT,m] = AnalyticalInput()
%This function returns the tensor of inertia TensT and the ...
    coordinates of
%the center of mass of a cube with an spherical hole in it with ...
    respect to
%the geometrical centre of the cube which we consider to be our ...
    origin
dens=2700;%Density of the Cube [Kg/m^3)
LC=0.1;%Longitude of cube [m]
REsf=LC/4;%the sphere hole will have a radious 4 times smaller ...
    than the cube length
CEsf=[LC/2,LC/2,LC/2];%Cartesian coordinates of the center of ...
    the sphere
ICube=eye(3,3)*((LC^2)/6)*(dens*LC^3);%Inertia tensor of our cube
IEsf=eye(3,3)*(2/5)*(REsf^2)*(4*pi*REsf^3/3)*dens...
    +(4*pi*REsf^3/4)*dens*[CEsf(2)^2+CEsf(3)^2, ...
        -CEsf(1)*CEsf(2), -CEsf(1)*CEsf(3);
                -CEsf(1)*CEsf(2), CEsf(1)^2+CEsf(3)^2, ...
                    -CEsf(2)*CEsf(3);
                -CEsf(1)*CEsf(3), -CEsf(2)*CEsf(3), ...
                    CEsf(1)^2+CEsf(2)^2]*(4*pi*REsf^3/3)*dens;%Inertia ...
                    tensor of the sphere displaced
TensT=ICube-IEsf;%Tensor of the cube with the sphere
MEsf=(4*pi*REsf^3/4)*dens;%Mass of our sphere
MCube=LC^3*dens;%Mass cube
CoordT=-1/(MCube-MEsf)*MEsf*CEsf;%Calculus of the coordinates of ...
    the whole thing
m=MCube-MEsf;
```

Listing C.2: Period introduction function.

```
function PeriodV=AskPeriod(InitialStatement,FollowingStatement)
 %This function as the user for all the periods of a section of the
 %experiment and returns it in vector form
 %The Inital Statement and the FollowingStatement are the ...
    questions asked
 %to the user and are both strings
  fprintf(InitialStatement);
  %The program tells the user what tipe of data should he/she input
  PeriodV(1)=-1;
  %The program ask the user for several periods in order to ...
     account for all
  %the experiments
  %When the user has introduced all of the periods he/she writes 0;
  i=1;
  finished=1;
    while (finished≠0)
        realpos=0;
        dev=0;
        while((realpos==0)&&(dev==0))
```

```
18              PeriodV(i)=input(FollowingStatement);%Store all the ...
                    periods in a vector
19              real=isreal(PeriodV(i));
20              pos=(PeriodV(i)≥0);
21              realpos=real*pos;
22              if (length(PeriodV)>3)&&(realpos==1)&&(abs(PeriodV(i)...
23                      -mean(PeriodV))>std(PeriodV))&&(PeriodV(i)≠0)
24              cdev=abs(PeriodV(i)-mean(PeriodV));
25              fprintf('Warning the current value deviation %d is ...
                    above the standard deviation %d ...
                    \n',cdev,std(PeriodV));
26              dev=input('Press 0 if you want to repeat this ...
                    variable, press any key if you do not \n');
27              if (dev≠0)
28                  PeriodV(i)=mean(PeriodV(1:(length(PeriodV)-1)));
29              end
30              end
31          finished=PeriodV(i);%Check if the user is done
32          i=i+1;
33          end
34      end
```

Listing C.3: Component computing.

```
1  function [Ip]=CalComp(T0,b)
2  %for i=1:length(T0)
3  Ip=(T0/(2*pi)).^2*b;
4  %end
```

Listing C.4: Main loop.

```
1  clc;
2  clear all;
3  close all;
4  rng('shuffle','V5normal' );
5  repeat =1;
6  while (repeat==1)
7      sim=2;
8      while ((sim≠0)&&(sim≠1))||(¬isfloat(sim))
9          sim=input('Would you like to simulate the experiment(1) ...
               or to input the data from a real measure(0): ');
10     end
11     if (sim==0)
12         %In this section the program asks the user for all the ...
               values of the
13         %experiment performed in real life
14         %The values of the calibration
15         repeat=0;
16         while (repeat==0)
17             b=input('Please enter the value of b[N m^2]: \n');
18             fprintf('Please measure the period of the pendulum ...
                   without anything placed on it \n');
19             fprintf('Make several measurements in order to ...
                   lessen the effect of the random error(We recomend ...
                   4) \n')
```

```
20              T0=AskPeriod('Write the periods of the ...
                    calibration[S]: \n','Input a period, if done ...
                    input 0: \n');
21              m=input('Please write the mass of the satellite[Kg] ...
                    \n');
22              repeat=input('In the event of needing to repeat this ...
                    section please write 0: \n');
23          end
24          repeat=0;
25          while (repeat==0)
26          %Ask the user for the values needed for Block 1
27              fprintf('BLOCK 1: Principal moments \n ');
28              %Measure of Izz
29              fprintf('Measure of Izz(Experiment 1.1) \n ');
30              fprintf('Please place the satellite on the platform ...
                    with its Z axis pointing up \n' );
31              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the Y axis(Experiment ...
                    1.1.1) \n');
32              dy1=input('Input the displacement of the satellite ...
                    in its vertical axis[m] \n');
33              T1=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced vertically [s] ...
                    \n','Input a period, if done input 0: \n');
34              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the X axis(Experiment ...
                    1.1.2) \n');
35              dx1=input('Input the displacement of the satellite ...
                    in its horizontal axis[m] \n');
36              T2=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced horizontally [s] ...
                    \n','Input a period, if done input 0: \n');
37              fprintf('Make sure the chosen coordinate origin is ...
                    contained within the rotation axis and perform ...
                    measurements(Experiment 1.1.3) \n');
38              T3=AskPeriod('Write the periods of the experiment ...
                    with the satellite on the center of rotation ...
                    \n','Input a period, if done input 0: \n');
39              %We now have the measures of the periods on three ...
                    vectors
40              %We create a matrix containing them and, if the ...
                    vectors are of
41              %different lengths we fill the matrix with the mean
42              columns=max([length(T1),length(T2),length(T3)]);
43              Tmat1=[T1,ones(1,(columns-length(T1)))*mean(T1);
44                   T2,ones(1,(columns-length(T2)))*mean(T2);
45                   T3,ones(1,(columns-length(T3)))*mean(T3)];
46              repeat=input('In the event of needing to repeat this ...
                    section please write 0: \n');
47          end
48          repeat=0;
49          while (repeat==0)
50              %The programs repeats the previous questions for the ...
                    Ixx and the Iyy
51              fprintf('Measure of Iyy (Experiment 1.2) \n ');
52              fprintf('Please place the satellite on the platform ...
```

```
                with its Y axis pointing up \n' );
53              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the Z axis(Experiment ...
                    1.2.1) \n');
54              dz2=input('Input the displacement of the satellite ...
                    in its vertical axis[m] \n');
55              T1=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced vertically [s] ...
                    \n','Input a period, if done input 0: \n');
56              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the X axis(Experiment ...
                    1.2.2) \n');
57              dx2=input('Input the displacement of the satellite ...
                    in its horizontal axis[m] \n');
58              T2=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced horizontally [s] ...
                    \n','Input a period, if done input 0: \n');
59              fprintf('Make sure the chosen coordinate origin is ...
                    contained within the rotation axis and perform ...
                    measurements(Experiment 1.2.3) \n');
60              T3=AskPeriod('Write the periods of the experiment ...
                    with the satellite on the center of rotation ...
                    \n','Input a period, if done input 0: \n');
61              Tmat2=[T1,ones(1,(columns-length(T1)))*mean(T1);
62                   T2,ones(1,(columns-length(T2)))*mean(T2);
63                   T3,ones(1,(columns-length(T3)))*mean(T3)];
64              repeat=input('In the event of needing to repeat this ...
                    section please write 0: \n');
65          end
66          repeat=0;
67          while (repeat==0)
68              fprintf('Measure of Ixx (Experiment 1.3) \n ');
69              fprintf('Please place the satellite on the platform ...
                    with its X axis pointing up \n' );
70              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the Y axis(Experiment ...
                    1.3.1) \n');
71              dy3=input('Input the displacement of the satellite ...
                    in its vertical axis[m] \n');
72              T1=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced vertically [s] ...
                    \n','Input a period, if done input 0: \n');
73              fprintf('Move the satellite a certain distance from ...
                    the rotation center making sure the rotation ...
                    center is contained within the Z axis(Experiment ...
                    1.3.2) \n');
74              dz3=input('Input the displacement of the satellite ...
                    in its horizontal axis[m] \n');
75              T2=AskPeriod('Write the periods of the experiment ...
                    with the satellite displaced horizontally [s] ...
                    \n','Input a period, if done input 0: \n');
76              fprintf('Make sure the chosen coordinate origin is ...
                    contained within the rotation axis and perform ...
                    measurements(Experiment 1.3.3) \n');
77              T3=AskPeriod('Write the periods of the experiment ...
```

```
                with the satellite on the center of rotation ...
                \n','Input a period, if done input 0: \n');
78              Tmat3=[T1,ones(1,(columns-length(T1)))*mean(T1);
79                  T2,ones(1,(columns-length(T2)))*mean(T2);
80                  T3,ones(1,(columns-length(T3)))*mean(T3)];
81             repeat=input('In the event of needing to repeat this ...
                  section please write 0: \n');
82          end
83          repeat=0;
84          %Introduce data for Block 2
85          while (repeat==0)
86              %Now we ask the user to place the support for the ...
                    cubesat and we
87              %perform another calibration
88              fprintf('Please place the suport of 45   on the ...
                    platform. \n');
89              T0sup=AskPeriod('Write the periods of the pendulum ...
                    with ONLY the support [s]. \n ','Input a period, ...
                    if done input 0: \n');
90
91              %Ask the user for the values needed for Block 2
92              fprintf('BLOCK 2: Moments outside the diagonal. \n ');
93              fprintf('Measure of Izy (Experiment 2.1)');
94              fprintf('Place the satellite on the support with the ...
                    X axis parallel to the platform. \n Make sure ...
                    that the selected coordinate origin is contained ...
                    within the rotation axis. \n');
95              T1exp=AskPeriod('Write the periods of the satellite ...
                    on the support. ','Input a period, if done input ...
                    0: \n');
96              fprintf('Measure of Izx (Experiment 2.2)');
97              fprintf('Place the satellite on the support with the ...
                    Y axis parallel to the platform. \n Make sure ...
                    that the selected coordinate origin is contained ...
                    within the rotation axis. \n');
98              T2exp=AskPeriod('Write the periods of the satellite ...
                    on the support. ','Input a period, if done input ...
                    0: \n');
99              fprintf('Measure of Iyx (Experiment 2.3)');
100             fprintf('Place the satellite on the support with the ...
                    Z axis parallel to the platform. \n Make sure ...
                    that the selected coordinate origin is contained ...
                    within the rotation axis. \n');
101             T3exp=AskPeriod('Write the periods of the satellite ...
                    on the support. ','Input a period, if done input ...
                    0: \n');
102             repeat=input('In the event of needing to repeat this ...
                    section please write 0: \n');
103         end
104         repeat=0;
105     end
106 %Simulation variables
107     if (sim==1)
108         options=3;
109         while ((options≠0)&&(options≠1)&&(options≠2))
110             options=input('Would you like to enter the tensor ...
                    data manually 0, read it from a file 1 or use the ...
                    default example 2: \n');
```

```
111
112         end
113         Initial=2;
114         prec=-5;%Order of the precision for the time measurement ...
                equipment
115         precm=-5;%Order of the precision of the mass measurement
116         siser=10^-7;%The sistematic error of the time ...
                measurement equipment
117         siserm=10^-7;%The sistematic error of the mass measurement
118         b=0.5;%Torsion Coefficient N*m^2
119         precb=-7;%Order of the precision for the torsion coefficient
120         siserb=10^-9;%Sistematic error for the torsion coefficient
121         Iempty=0.38825;%The moment of the empty pendulum
122         Isup=0.589;%The moment of the pendulum with the support
123         dy1=0.1;%The displaced distance in Block 1 section 1
124         dx1=0.1;%The displaced distance in Block 1 section 1
125         dz2=0.1;%The displaced distance in Block 1 section 2
126         dx2=0.1;%The displaced distance in Block 1 section 2
127         dy3=0.1;%The displaced distance in Block 1 section 3
128         dz3=0.1;%The displaced distance in Block 1 section 3
129         ni=4;%Number of times each separate period is measured
130         while ((Initial≠0)&&(Initial≠1))
131             fprintf('The default experiment parameters are \n');
132             fprintf('Precision of the timing equipment[s]: %d ...
                    \n',(10^prec));
133             fprintf('Sistematic error of the timing ...
                    equipment[s]: %d \n',(siser));
134             fprintf('Precision of the mass measurement ...
                    equipment[Kg]: %d \n',(10^precm));
135             fprintf('Sistematic error of the mass measurement ...
                    equipment[Kg]: %d \n',(siserm));
136             fprintf('The torsion coefficient is: %d N*m^2 \n ',b);
137             fprintf('Precision of the torsion coefficient ...
                    measurement equipment[N*m^2]: %d \n',(10^precb));
138             fprintf('Sistematic error of the torsion coefficient ...
                    equipment[N*m^2]: %d \n',(siserb));
139             fprintf('The distance moved in each section of Block ...
                    1 has been[m]: \n');
140             fprintf('Section 1: dy= %d dx= %d \n',dy1,dx1);
141             fprintf('Section 2: dz= %d dx= %d \n',dz2,dx2);
142             fprintf('Section 3: dy= %d dz= %d \n',dy3,dz3);
143
144             Initial=input('Would you like to manually change the ...
                    experiment parameters 0,  or use the default ones ...
                    1: \n');
145
146         end
147         if (Initial==1)
148             fprintf('You have selected the default experiment ...
                    parameters. \n');
149         elseif(Initial==0)
150             b=input('Please enter the value of b[N m^2]: \n');
151             precb=input('Please enter the precission of b[N ...
                    m^2]: \n');
152             precb=log10(precb);
153             siserb=input('Please enter the sistematic error of ...
                    b[N m^2]: \n');
154             prec=input('Please input the precission of the ...
```

```
                timing equipment in seconds: \n');
155             prec=log10(prec);
156             siser=input('Please input the sistematic error of ...
                    the timing equipment[S]: \n');
157             precm=input('Please enter the precission of m[Kg]: \n');
158             precm=log10(precm);
159             siserm=input('Please enter the sistematic error of ...
                    the mass measuremt equipment[Kg]: \n');
160             Iempty=input('Please enter the moment of the empty ...
                    pendulum: \n');
161             Isup=input('Please enter the moment of the pendulum ...
                    with the support: \n');
162             ni=input('Please enter the number of times each ...
                    separate period is measured: \n');
163         end
164         br=b+(-1+2*rand)*10^precb+siserb;
165         T1=sqrt(Iempty/br)*2*pi;
166         T0=T1*ones(ni,1)+(-1+2*rand(ni,1))*10^prec+siser;% ...
                Period of the empty pendulum [s]
167         T1sup=sqrt(Isup/br)*2*pi;
168         T0sup=T1sup*ones(ni,1)+(-1+2*rand(ni,1))*10^prec+siser;
169
170     %     TensT=[0.0045,0,0;
171     %             0,0.0045,0
172     %             0,0,0.0045];%Tensor anal tico del experimento ...
            respecto al sistema de referencia elegido
173     %     CoordT=[0,0,0];%Coordenadas X,Y,Z del centro de masas
174         if (options==2)
175         [TensT,CoordT,m]=AnalyticalInput();
176         elseif(options==1)
177         [TensT,CoordT,m]=ReadFile();
178         elseif(options==0)
179         [TensT,CoordT,m]=ManualInput();
180         end
181         mr=m+(-1+2*rand)*10^precm+siserm;
182         TensTCM=TensT-mr*[CoordT(2)^2+CoordT(3)^2, ...
                -CoordT(1)*CoordT(2), -CoordT(1)*CoordT(3);
183                          -CoordT(1)*CoordT(2), ...
                             CoordT(1)^2+CoordT(3)^2, ...
                             -CoordT(2)*CoordT(3);
184                          -CoordT(1)*CoordT(3), ...
                             -CoordT(2)*CoordT(3),CoordT(1)^2+CoordT(2)^2];
185     end
186     %Calibrations
187     T0m=mean(T0);
188     I0m=CalComp(T0m,b);
189     T0supm=mean(T0sup);
190     I0supm=CalComp(T0supm,b);
191     %Initialize the two matrixes in which we will store our results
192     %Tensor with respect to the CM
193
194     CMtens=zeros(3,3);
195     %Tensor with respect to the CG
196     CGtens=zeros(3,3);
197     %%
198     %BLOCK 1 Determination of the principal moments
199     %Calculation of the Izz cubesat
200     %Experimental results are stored in the matrix Tmat1 of size 3x4
```

```
    if (sim==1)
T1=sqrt((Iempty+TensTCM(3,3)+((dy1-CoordT(2))^2+...
    CoordT(1)^2)*mr)/br)*2*pi;
T1sim=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
T2=sqrt((Iempty+TensTCM(3,3)+((dx1+CoordT(1))^2+...
    CoordT(2)^2)*mr)/br)*2*pi;
T2sim=T2*ones(1,ni)+rand(1,ni)*10^prec+siser;
T3=sqrt((Iempty+TensTCM(3,3))/br)*2*pi;
T3sim=T3*ones(1,ni)+rand(1,ni)*10^prec+siser;
Tmat1=[T1sim;T2sim;T3sim];
        %We create the matrix Tmat1
    end
    I1=CalComp(mean(Tmat1(1,:)),b)-I0m;
    I2=CalComp(mean(Tmat1(2,:)),b)-I0m;
    I3=CalComp(mean(Tmat1(3,:)),b)-I0m;
    %desviaciones[m]
    dx=dx1;
    dy=dy1;
    X01=(I2-I3)/(m*dx*2)-dx/2;
    Y01=((I1-I3)/(m*dy*2)-dy/2)*(-1);
    Izz=I3-m*(X01^2+Y01^2);
    CMtens(3,3)=(Izz);
    CGtens(3,3)=(I3);

    %Calcular Iyy cubesat rotando 90   en el eje X
    %Guardamos en una matriz 4x3 los resultados experimentales
    if (sim==1)
T1=sqrt((Iempty+TensTCM(2,2)+((dz2+CoordT(3))^2+...
    CoordT(1)^2)*mr)/br)*2*pi;
T1sim=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
T2=sqrt((Iempty+TensTCM(2,2)+((dx2+CoordT(1))^2+...
    CoordT(3)^2)*mr)/br)*2*pi;
T2sim=T2*ones(1,ni)+rand(1,ni)*10^prec+siser;
T3=sqrt((Iempty+TensTCM(2,2))/br)*2*pi;
T3sim=T3*ones(1,ni)+rand(1,ni)*10^prec+siser;
Tmat2=[T1sim;T2sim;T3sim];
        %De momento ultilizaremos estos c lculos
    end
    I1=CalComp(mean(Tmat2(1,:)),b)-I0m;
    I2=CalComp(mean(Tmat2(2,:)),b)-I0m;
    I3=CalComp(mean(Tmat2(3,:)),b)-I0m;%De momento ultilizaremos ...
       estos c lculos
    %desviaciones[m]
    dx=dx2;
    dz=dz2;
    Z02=(I1-I3)/(m*dz*2)-dz/2;
    X02=(I2-I3)/(m*dx*2)-dx/2;
    Iyy=I3-m*(X02^2+Z02^2);
    CMtens(2,2)=(Iyy);
    CGtens(2,2)=(I3);

    %Calcular Ixx cubesat rotando 90   en el eje Y
    %Guardamos en una matriz 4x3 los resultados experimentales
    if (sim==1)
T1=sqrt((Iempty+TensTCM(1,1)+((dy3-CoordT(2))^2+...
    CoordT(3)^2)*mr)/br)*2*pi;
T1sim=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
T2=sqrt((Iempty+TensTCM(1,1)+((dz3-CoordT(3))^2+...
```

```
    CoordT(2)^2)*mr)/br)*2*pi;
T2sim=T2*ones(1,ni)+rand(1,ni)*10^prec+siser;
T3=sqrt((Iempty+TensTCM(1,1))/br)*2*pi;
T3sim=T3*ones(1,ni)+rand(1,ni)*10^prec+siser;
Tmat3=[T1sim;T2sim;T3sim];
%De momento ultilizaremos estos c lculos
    end
    I1=CalComp(mean(Tmat3(1,:)),b)-I0m;
    I2=CalComp(mean(Tmat3(2,:)),b)-I0m;
    I3=CalComp(mean(Tmat3(3,:)),b)-I0m;
    %desviaciones[m]

    dy=dy3;
    dz=dz3;
    Y03=((I1-I3)/(m*dy*2)-dy/2)*(-1);
    Z03=((I2-I3)/(m*dz*2)-dz/2)*(-1);
    Ixx=I3-m*(Y03^2+Z03^2);
    CMtens(1,1)=(Ixx);
    CGtens(1,1)=(I3);
    %%
    %BLOCK 2;
    %DETERMINATION OF COMPONENTS OUTSIDE THE DIAGONAL
    %ONLY FOR COMPONENTS WITH RESPECT TO THE GEOMETRIC CENTER
    %angles of rotation 45 deg respect to X
    if (sim==1)
        Iprot=RotMat(0,0,pi/4,TensT);
        T1=sqrt((Iprot(3,3)+Isup)/br)*2*pi;
        T1exp=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
    end
    I1=CalComp(mean(T1exp),b)-I0supm;
    CGtens(2,3)=(I1-CGtens(1,1)/2-CGtens(3,3)/2)*(-1);
    CGtens(3,2)=CGtens(2,3);
    %angles of rotation 45 deg respect to Y
    if (sim==1)
        Iprot=RotMat(0,pi/4,0,TensT);
        T1=sqrt((Iprot(3,3)+Isup)/br)*2*pi;
        T2exp=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
    end
    I1=CalComp(mean(T2exp),b)-I0supm;
    CGtens(1,3)=I1-CGtens(1,1)/2-CGtens(3,3)/2;
    CGtens(3,1)=CGtens(1,3);
    %angles of rotation 90 deg respect to X & 45 deg respect to Y
    if (sim==1)
        Iprot=RotMat(pi/2,pi/4,0,TensT);
        T1=sqrt((Iprot(3,3)+Isup)/br)*2*pi;
        T3exp=T1*ones(1,ni)+rand(1,ni)*10^prec+siser;
    end
    I1=CalComp(mean(T3exp),b)-I0supm;
    CGtens(2,1)=I1-CGtens(1,1)/2-CGtens(2,2)/2;
    CGtens(1,2)=CGtens(2,1);
    %%
    %FINAL CALCULATIONS AND ERRORS
    Xcm=mean([X01,X02]);
    Ycm=mean([Y01,Y03]);
    Zcm=mean([Z02,Z03]);
    CoordV=[Xcm,Ycm,Zcm];
    Mat=ones(3,3)-eye(3);
    CMtens=CGtens-diag(diag(CGtens))-m*[0, -Xcm*Ycm, -Xcm*Zcm;
```

```
316                     -Xcm*Ycm, 0, -Ycm*Zcm;
317                     -Xcm*Zcm, -Ycm*Zcm, 0]+diag(diag(CMtens));
318     [VCG,DCG]=eig(CGtens);
319     [VCM,DCM]=eig(CMtens);
320     fprintf('These are the results of the program: \n');
321     fprintf('The calculated tensor with respect to the chosen ...
           coordinate origin has been[Kg*m^2]: \n');
322     disp(CGtens);
323     fprintf('The calculated tensor with respect to the center of ...
           mass has been[Kg*m^2]: \n');
324     disp(CMtens);
325     fprintf('The calculated coordinates for the center of mass ...
           have been [X,Y,Z]m: \n');
326     disp(CoordV);
327     fprintf('The principal axis of inertia with respect to the ...
           coordinate origin are: \n')
328     fprintf('1st vector: %d \n',VCG(:,1)');
329     fprintf('2nd vector: %d \n',VCG(:,2)');
330     fprintf('3rd vector: %d \n',VCG(:,3)');
331     fprintf('With its inertia tensor beeing [Kg*m^2]: \n');
332     disp(DCG);
333     fprintf('The principal axis of inertia with respect to the ...
           center of mass are: \n')
334     fprintf('1st vector: %d \n',VCM(:,1)');
335     fprintf('2nd vector: %d \n',VCM(:,2)');
336     fprintf('3rd vector: %d \n',VCM(:,3)');
337     fprintf('With its inertia tensor beeing [Kg*m^2]: \n');
338     disp(DCM);
339     if (sim==1)
340         ErrorMaTCG=((TensT-CGtens)./(CGtens))*100;
341         fprintf('The relative error, in percentage, the tensor ...
              with respect to the coordinate origin has been: \n');
342         disp(ErrorMaTCG);
343         ErrorMaTCM=((TensTCM-CMtens)./(CMtens))*100;
344         fprintf('The relative error, in percentage, the tensor ...
              with respect to the center of mass has been: \n');
345         disp(ErrorMaTCM);
346         ErrorCM=((CoordT-CoordV)./CoordT)*100;
347         fprintf('The relative error, in percentage, of the ...
              calculation of the center of mass coordinates has ...
              been: \n');
348         disp(ErrorCM);
349         absErrorMaTCG=abs(TensT-CGtens);
350         fprintf('The absolute error in the tensor with respect ...
              to the coordinate origin has been: \n');
351         disp(absErrorMaTCG);
352         absErrorMaTCM=abs(TensTCM-CMtens);
353         fprintf('The absolute error in the tensor with respect ...
              to the center of mass has been: \n');
354         disp(absErrorMaTCM);
355         absErrorCM=abs(CoordT-CoordV);
356         fprintf('The absolute error in the calculation of the ...
              center of mass coordinates has been: \n');
357         disp(absErrorCM);
358     end
359     tensordisp=input('Enter 1 if you want to know the tensor ...
           with respect to a new point in the chosen coordinate ...
           system: \n');
```

```
360     if (tensordisp==1)
361         Newpoint=input('Please enter the coordinates of the new ...
               point separated by commas within square brackets ...
               [X,Y,Z] in meters: \n');
362         NewVector=CoordV-Newpoint;
363         NewTens=CMtens+m*[NewVector(2)^2+NewVector(3)^2, ...
               -NewVector(1)*NewVector(2), -NewVector(1)*NewVector(3);
364                         -NewVector(1)*NewVector(2), ...
                               NewVector(1)^2+NewVector(3)^2, ...
                               -NewVector(2)*NewVector(3);
365                         -NewVector(1)*NewVector(3), ...
                               -NewVector(2)*NewVector(3), ...
                               NewVector(2)^2+NewVector(1)^2];
366         fprintf('The tensor with respect to the point %d,%d,%d ...
               is [Kg*m^2]: \n',Newpoint(1),Newpoint(2),Newpoint(3));
367         disp(NewTens);
368         tensordisp1=input('Enter 1 if you want to know the ...
               tensor with respect to a rotated set of axis with ...
               respect to the new point: \n');
369         if(tensordisp1==1)
370             NewAngles=input('Please enter the new angles ...
                   separated by commas within square brackets ...
                   [alpha,beta,gamma] in degrees: \n');
371             NewAngles=NewAngles*pi/180;
372             NewTens=RotMat(NewAngles(1),NewAngles(2),NewAngles(3),NewTens);
373             fprintf('The tensor with respect to the point %d  ...
                   and rotated %d rad is [Kg*m^2]: ...
                   \n',(Newpoint),(NewAngles));
374             disp(NewTens);
375         end
376         [Vnew,Dnew]=eig(NewTens);
377         fprintf('The principal axis of inertia with respect to ...
               the new reference system are: \n')
378         fprintf('1st vector: %d \n',Vnew(:,1)');
379         fprintf('2nd vector: %d \n',Vnew(:,2)');
380         fprintf('3rd vector: %d \n',Vnew(:,3)');
381         fprintf('With its inertia tensor beeing [Kg*m^2]: \n');
382         disp(Dnew);
383     end
384
385     write=input('Enter 1 if you want to save the results in a ...
           .txt file enter any key if you do not: \n');
386     if (write==1)
387         WriteFile(CGtens,CMtens,CoordV,m);
388     end
```

Listing C.5: Manual input of tensor data.

```
1  function [TensT,CoordT,m]=ManualInput()
2  %The function asks the user for the manual introduction of the ...
       inertia
3  %tensor.
4  sim=0;
5  while (sim==0)
6  fprintf('Please make sure that the tensor introduced is with ...
       respect to the chosen coordinate center: \n');
7  fprintf('Please enter the requested tensor component in kg*m^2. ...
```

```
    \n');
for n=1:3
    for i=1:3
        fprintf('Please enter the component %d,%d of the matrix: ...
            \n ',n,i);
        realpos=0;
        while realpos==0
            Moment=input('Moment= ');
            real=isreal(Moment);
            pos=(Moment≥0);
            realpos=real*pos;
        end
        TensT(n,i)=Moment;
    end
end
sim=issymmetric(TensT);
if (sim==0)
    fprintf('WARNING: The introduced Matrix is not symmetric, ...
        please enter the matrix again');
end
end
fprintf('Please enter the coordinates of the center of mass ...
    making sure the coordinate sistem used corresponds to the one ...
    used to input the matrix \n');
CoordT(1)=input('Please enter the X coordinate in meters: \n');
CoordT(2)=input('Please enter the Y coordinate in meters: \n');
CoordT(3)=input('Please enter the z coordinate in meters: \n');
m=('Please enter the mass of the satellite in KG: \n');

end
```

Listing C.6: Function to read tensor data of file.

```
function [TensT,CoordT,m]=ReadFile()
%The function reads the data from a text file
%The data is stored in a 7X3 matrix containing de tensor with ...
    respect to
%the center of mass the tensor with respect to the coordinate ...
    center and de
%coordinates XYZ of the center of mass.
```

Listing C.7: Fuction to compute rotated tensor.

```
function [RotM] = RotMat(alpha,beta,gamma,M)
%M 3 dimensional tensor[kg*m^2]
%alpha, beta, gamma rotation angles [rad]
%RotV 3 dimensional tensor[Kg*m^2]
%This function calculates the new coordinates of the tensor M in ...
    a system
%rotated by the specified angles and returns them in RotM
%1st step rotational matrix
EulM=[cos(alpha)*cos(beta), ...
    cos(alpha)*sin(beta)*sin(gamma)-sin(alpha)*cos(gamma), ...
    cos(alpha)*sin(beta)*cos(gamma)+sin(alpha)*sin(gamma)
     sin(alpha)*cos(beta), ...
        cos(alpha)*sin(beta)*sin(gamma)+cos(alpha)*cos(gamma), ...
```

```
          sin(alpha)*sin(beta)*cos(gamma)-cos(alpha)*sin(gamma)
    -sin(beta), cos(beta)*sin(gamma), cos(beta)*cos(gamma)
    ];
RotM=EulM*M*EulM';
```

Listing C.8: Function to write tensor data of file.

```
function WriteFile(CGtens,CMtens,CoordV,m)
%this function saves the results of the experiment in a .txt file
savematrix=[CGtens;CMtens;CoordV;[m,0,0]];
[file,path]=uiputfile('.txt');
writematrix(savematrix,[path,file]);
```